Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Deutschland

Studienarbeit Nr. 2424

Analyse und Prognose von Umweltdaten in Geschäftsprozessen

Lazar Davidkov

Studiengang:	Informatik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	M.Sc. Wirt.-Inf. Alexander Nowak
begonnen am:	16.04.2013
beendet am:	11.10.2013
CR-Klassifikation:	H.4.1

Inhaltsverzeichnis

1 Einleitung 1
2 Definitionen 3
2.1 Business Process / Workflow 3
2.2 Green IT 5
2.2.1 Green Organisations 5
2.2.2 Green Technology 7
2.2.3 Green Processes 7
2.3 Business Intelligence Werkzeuge 8
2.3.1 Data Warehouse 8
2.3.2 ETL-Prozesse 9
2.3.3 Dashboard 10
3 Konzept 11
3.1 Zielbestimmung 11
3.2 Funktionale Anforderungen 11
3.3 Nicht-funktionale Anforderungen 12
4 Design 14
4.1 Java EE Architektur 14
4.2 KEIDA – Überblick 16
4.3 KEIDA – Design 17
4.4 KEIDA – Architektur 18
4.4.1 Serverseitige Architektur 18
4.4.2 Clientseitige Architektur 19
5 Implementierung 21
5.1 SpringMVC und jQuery 21
5.2 Konfiguration 22
5.3 Controllerschicht (Controller Layer) 24
5.4 Präsentationsschicht (View Layer) 26
5.5 Modellschicht (Model Layer) 28
5.6 GUI Implementierung 32
6 Anwendungsfall: Optimieren eines Prozesses in KEIDA 33
7 Zusammenfassung 39
8 Anhang A – KEIDA Konfiguration 40
9 Anhang B – KEIDA Klassenstruktur 44
Literaturverzeichnis 49

Abkürzungsverzeichnis

API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
BI	Business Intelligence
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Management Notation
DAO	Data Access Object
DWH	Data Warehouse
ETL	Extract, Transform, Load
GUI	Graphical User Interface
GW	Gigawatt
JSON	JavaScript Object Notation
KEIDA	Key Ecological Indicators Dashboard
KPI	Key Performance Indicator
MVC	Model, View, Controller
SOA	Service Oriented Architecture
SVG	Scalable Vector Graphics
WfMC	Workflow Management Coalition
WS	Web Service
WSDL	Web Service Description Language

Abbildungsverzeichnis

Abb. 2.1 IBM Business Process Manager

Abb. 2.2 The devil's pentagon

Abb. 4.1 MVC Entwurfsmuster

Abb. 4.2 Architektur einer Webanwendung

Abb. 4.3 KEIDA Dashboard

Abb. 4.4 KEIDA Visualisierungswerkzeug

Abb. 4.5 KEIDA Architektur

Abb. 4.6 Bearbeitung einer Anfrage in KEIDA

Abb. 5.1 DispatcherServlet workflow

Abb. 5.2 Servlet- und Servlet-Mapping-Definitionen in web.xml

Abb. 5.3 Importieren von Konfigurationsdateien in spring-servlet.xml

Abb. 5.4 Auszug aus der spring-servlet.xml -Datei

Abb. 5.5 Tiles 2.2 Konfiguration

Abb. 5.6 MySQL Konfiguration

Abb. 5.7 ViproWS Konfiguration in keida-viproWS-konfiguration.xml

Abb. 5.8 Handler Mapping

Abb. 5.9 Controllerklassen

Abb. 5.10 Mapping einer Controller Klasse

Abb. 5.11 Mapping einer AJAX Anfrage

Abb. 5.12 View Konfiguration in views.xml

Abb. 5.13 Ausschnitt aus index.jsp

Abb. 5.14 Auflösung des Namens eines Views

Abb. 5.15 Definition der Startseite des Dashboards

Abb. 5.16 KEIDA Process page mock-up

Abb. 5.17 processView()-Methode der ProcessPageController-Klasse

Abb. 5.18 getViewPage-Methode der PPC-Klasse

Abb. 5.19 chartMonthlyData()-Methode der Klasse ChartServiceWorker

Abb. 5.20 Ausschnitt aus der Klasse ChartJDBCTemplate

Abb. 6.1 KEIDA Dashboard

Abb. 6.2 PurchaseOrderResselerProcess Dashboard

Abb. 6.3 Verbrauch grüner Energie

Abb. 6.4 Durchschnittlicher Anteil der grünen Energie bei einem Prozessdurchlauf

Abb. 6.5 Konfigurationsassistent

Abb. 6.6 Visualisierung des PurchaseOrder Prozesses

Abb. 6.7 Vergleich von zwei Visualisierungen

Abb. 8.1 MySQL Konfiguration in keida-db.xml

Abb. 8.2 Homepage Konfiguration in der keida-homepage-configuration.xml-Datei

Abb. 8.3 ViproWS Konfiguration

Abb. 8.4 ColorMap Konfiguration

Abb. 9.1 showDashboard() - Methode der Klasse HomepageController

Abb. 9.2 getDayValue()-Methode der Klasse chartJDBCTemplate

1 Einleitung

Heute ist es kaum denkbar, dass ein Unternehmen existieren kann, ohne eine eigene IT zu haben oder IT-Dienstleistungen zu beziehen. Im Jahr 2020 wird die Mehrzahl der Geschäftsprozesse in der Wirtschaft mit Hilfe von IT-Systemen betrieben werden [Eg13]. Diese Systeme (Hardware und darauf laufende Software) führen zu einem enormen Energieverbrauch in den Unternehmen. Google z.B. verbraucht kontinuierlich 260 Millionen Watt, was einem Viertel der Energie eines Atomkraftwerks entspricht oder dem Strom, der benötigt wird, um eine Stadt mit 200 000 Haushalten zu versorgen [Kr13]. Nur die Rechenzentren in Deutschland verbrauchen die Energie, die der Leistung von vier mittleren Kohlekraftwerken entspricht, nämlich 10 Milliarden kWh [KW10]. Bei der Produktion dieser Energie wird eine enorme Menge CO_2 in die Atmosphäre ausgestoßen. Dieser Energieverbrauch ist auch mit hohen Kosten verbunden. Das führte in den letzten Jahren dazu, dass das Management in vielen Unternehmen mit dem Problem konfrontiert wurde (und ist immer noch), wie dieser Energieverbrauch verringert werden kann, um die steigenden Kosten einzudampfen. Lösungen gibt es viele und alle sind direkt oder indirekt mit einem nachhaltigen Umgang mit den knappen Ressourcen verbunden, die zu Verfügung stehen und bei der Energiegewinnung eine Rolle spielt. Als Nebeneffekt tragen die Unternehmen aktiv dazu bei, die Umwelt so wenig wie möglich zu belasten. Die Unternehmen sollen heute nicht nach dem “single bottom line”-Prinzip geführt werden, sondern nach dem “tripple bottom line”-Prinzip [BS12]. Dieses enthält nicht nur den Erlös am Ende des Geschäftsjahres, sondern ist auf die Menschen und die Umwelt ausgerichtet. Diese Diskussionen und Überlegungen führten 2008 zur Formulierung des Begriffs “Green IT”. Green IT beruht auf zwei Ansätzen: Wie kann die IT zum nachhaltigen Umgang mit den begrenzten Ressourcen beitragen und wie lässt sich die IT nachhaltiger betreiben [LN11].

Die angestoßene Diskussion um die Nachhaltigkeit beim Umgang mit den begrenzten Ressourcen zeigt langsam ihre Wirkung. So haben Branchenriesen wie Google und Facebook die Notwendigkeit von energiesparenden Rechenzentren sehr schnell erkannt und bauen diese im Norden (in Finnland, im Fall von Facebook), um die Rechner mit Meerwasser zu kühlen. Bei Google werden die Gebäude und die Hardware so konstruiert, dass diese am wenigsten Energie verbrauchen [Go13]. Die Notwendigkeit, weniger Energie zu verbrauchen, führt bei vielen Unternehmen zum Überdenken der ganzen IT-Strategie. Das deutsche Unternehmen Strato z.B. ersetzte nicht nur die Server mit energiesparenden Geräten, sondern auch die Software, die darauf läuft, wurde so umprogrammiert, dass die Algorithmen viel weniger CPU-Zyklen in Anspruch nehmen und dadurch viel weniger Strom bei den Berechnungen brauchen [BES10].

Das Beispiel von Strato zeigt, dass es wichtig ist, nicht nur die Hardware mit energiesparenden Lösungen zu ersetzen, sondern auch die Software so zu optimieren, dass diese den geringsten Energieverbrauch aufweist. KEI Framework stellt eine Lösung dar, die genau das anstrebt: eine Optimierung der Ökobilanz der Software, die hinter den Geschäftsprozessen eines Unternehmens steht. Das KEI Framework ist in der Lage, den Ablauf eines Prozesses im produktiven Einsatz zu überwachen, diesen in einer Testumgebung zu simulieren und Informationen zu dessen Energieverbrauch zu ermitteln. Diese Informationen sollen dazu dienen, den Prozess so zu gestalten, dass dieser möglichst effizient

und mit einem möglichst geringeren Energieverbrauch läuft. KEI Dashboard (KEIDA) ist das Front-End des KEI Frameworks und hat die Aufgabe, die Informationen, die vom KEI Framework gewonnen werden, graphisch darzustellen, um eine Analyse dieser Informationen zu ermöglichen. Diese Auswertung soll anhand von Grafiken erfolgen, die den Verlauf des Business Prozesses zeigen, sowie dessen einzelne Aktivitäten und den Beitrag der Aktivitäten zum gesamten Energieverbrauch. Mit Hilfe des Visualisierungswebservice ViproWS soll dann noch zusätzlich die Struktur des Prozesses graphisch dargestellt werden. Diese Darstellung basiert auf den Informationen über den Energieverbrauch jeder einzelnen Aktivität. Dadurch können die Benutzer, die einen Prozess analysieren, schnell nachzuvollziehen, wo sich die Problemzonen befinden. Durch Annotation sind die Benutzer in der Lage, eine zweite Darstellung des Prozesses zu bekommen, die sich mit der ersten vergleichen lässt. Das Ziel dabei ist ein direkter Vergleich der zwei Versionen des Prozesses.

Diese Studienarbeit gibt einen Überblick über die Technologien, die hinter dem KEI Framework stehen, und erläutert wie KEIDA konzipiert und implementiert ist.

Die vorliegende Studienarbeit gliedert sich neben Einleitung und Zusammenfassung in fünf Hauptteile. Nach der Einleitung folgt ein Überblick über die Begriffe, die eine große Rolle bei dem Aufbau des KEI Frameworks und KEIDA spielen. Eine kurze Einführung erläutert, was ein Geschäftsprozess ist und wie sich dieser mit Software realisieren lässt. Ein anderes wichtiges Thema, das besprochen wird, ist die Green IT und wie die grüne Technologie ihren Platz im Alltag findet. KEI Framework setzt auf Business Intelligence (BI) Konzepten. Aus diesem Grund wurden beim Aufbau des KEIDA viele Ideen aus diesem Bereich einbezogen. In diesem Zusammenhang werden auch Begriffe und wichtige Konzepte aus dem BI-Bereich eingeführt.

Funktionale und nicht-funktionale Anforderungen dienen als Basis jeder Software und sind ein wichtiger Teil jeder Dokumentation. Sie dienen als Vertrag zwischen Auftraggeber und Auftragnehmer und werden aufgrund ihrer Wichtigkeit in Kapitel 3 detailliert beschrieben. Anschließend wird das Design der Software erläutert und welche Konzepte dabei eigesetzt wurden. Es wird eine Übersicht über Java EE und die vier Tier Architektur angeboten, die diese ermöglicht und die als Grundlage aller modernen Webanwendungen dient. Die MVC Architekturstyle ist auch ein Bestandteil jeder Webanwendung und spielt eine große Rolle bei der Verteilung der Zuständigkeiten innerhalb der Software. KEIDA verfolgt diese Prinzipien und setzt auf die MVC-Architektur. Ein Überblick über die MVC-Architektur und wie diese in KEIDA eingesetzt wird gibt Kapitel 4.

In Kapitel 5 folgt die eigentliche Implementierung von KEIDA. Anhand von Beispielen, die aus dem Source Code der Software genommen wurden, werden der Aufbau und die Funktionsweise von KEIDA erläutert. Kapitel 6 veranschaulicht anhand eines Anwendungsfalls, wie sich die Software beim Optimieren eines Geschäftsprozesses einsetzen lässt. Anhang A enthält die Information, die bei einer Installation auf dem Server gebraucht wird, und Anhang B verschafft einen Überblick über die eigentliche Struktur der Java-Pakete und über die dazugehörigen Java-Klassen, die KEIDA bilden.

2 Definitionen

In dieser Studienarbeit werden Ansätze, die bei jeder BI-Lösung zu finden sind, eingesetzt. Wie schon in Kapitel 1 aufgeführt, liegt die Aufgabe darin, ein Dashboard aufzubauen, auf dem die im Data Warehouse abgelegten Informationen grafisch dargestellt und die Analyse dieser Informationen ermöglicht werden. Ein weiterer Schwerpunkt der Studienarbeit ist es, die Visualisierung anhand des bestehenden Visualisierungswebservice zu vereinfachen und benutzerfreundlich zu gestalten. Dabei wird dem Nutzer die Möglichkeit gegeben, die berechneten Werte mit neuen auszutauschen und die neue graphische Darstellung mit der alten zu vergleichen. Begriffe, die dabei eine große Bedeutung haben, sind Business Process, Business Process Management, Dashboard, Data Warehouse, Extract Transoform and Load (ETL) Prozesse und Key Ecological Indicators (KEI) und werden in diesem Kapitel näher erläutert.

2.1 Business Process / Workflow

Eine Definition des Business Process, auf Deutsch Gechäftsprozess, geben Davenport und Short [DS90]:

> "*A set of logically-related tasks performed to achieve defined business outcome.*"

und:

> *"1) Processes have customers; that is, processes have a defined business outcome, and there are recipients of the outcomes. Customers may be either internal or external to the firm; and*
>
> *2) They cross organizational boundaries; that is, normally that occurs across or between organizational subunits. Processes are generally independent of formal organizational structure.*"

Beispiele für Geschäftsprozesse sind die Erstellung eines Produkts, die Bestellung von Waren bei einem Lieferanten oder die Abwicklung einer Bestellung eines Kunden in einem Onlineshop.

Die Definition oben besagt, dass ein Prozess aus verschiedenen Aktivitäten besteht, wobei die Aktivitäten auch andere Prozesse sein können. Diese Aktivitäten werden in einer festgelegten zeitlichen und logischen Reihenfolge ausgeführt und so entsteht ein Workflow. Eine Definition des Begriffs Workflow findet man bei WfMC[1]: "the automation of a business process" [WMC]. Die Prozesse können auch sehr komplex sein und dadurch mehrere Workflows enthalten.

Geschäftsprozesse sind nicht etwas Statisches. Sie werden ständig geändert und an die neuen Anforderungen angepasst. Das ist nötig, damit diese immer wieder Wettbewerbsvorteile gegenüber der Konkurrenz liefern. Um diese Änderbarkeit zu vereinfachen, wird auf Business Prozess Management Systeme (BPM) gesetzt. BPM erleichtert das Design und die Entwicklung eines Prozesses, sowie seine kontinuierliche Verbesserung. Weitere wichtige

[1] The Workflow Management Coalition

Funktionen des BPMs sind die Reduktion von menschlichen Fehlern und die Verbesserung der Kommunikation zwischen den Stakeholdern (siehe Abb. 2.1). Die Stakeholder sind dann in der Lage, sich auf den Anforderungen ihrer Rollen zu konzentrieren [Rm11]. BPM hält die Administration dieses Änderungsprozesses einfacher. Die Weiterentwicklung bestehender Prozesse und derer einzelnen Aktivitäten und dadurch alle verbundenen Aufgaben wie Versionierung, Testen usw. werden vom BPM vereinfacht und aktiv unterstützt.

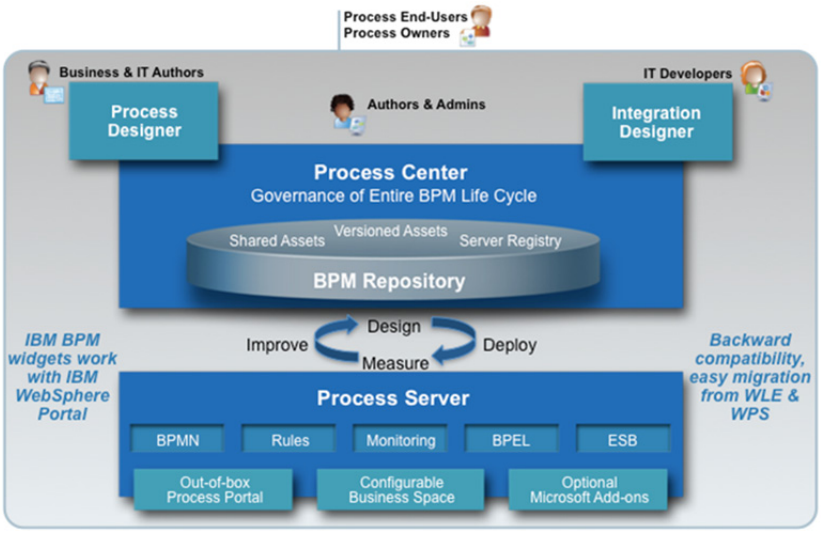

Abbildung 2.1: IBM Business Process Manager [Sk11]

Die Geschäftsprozesse die in dieser Studienarbeit betrachtet werden, sind mit Hilfe der Business Process Execution Language für Web Services (WS-BPEL) beschrieben. WS-BPEL ist xml-basiert und dient zur Orchestrierung von WS. Diese Web Services stellen die einzelnen Aktivitäten des Geschäftsprozesses dar. BPEL baut auf die WSDL auf und ermöglicht die Bereitstellung eines Prozesses selbst als WS, was den Aufbau von sehr komplexen Geschäftsprozessen vereinfacht. Die Webservices-Technologie definiert, wie Funktionalitäten über Internet Protokolle zugänglich gemacht werden können. WS-BPEL beschreibt, wie diese in eine logische Struktur zusammengeführt werden können, um eine spezifische Aufgabe zu lösen.

Die Orchestrierung der Web Services ist der zentrale Aspekt von WS-BPEL und stellt die Grundlage beim Aufbau von Service Oriented Architecture (SOA)-Lösungen dar. Das erfolgt in zwei Schritten:

- die Entwicklung und Veröffentlichung von Web Services
- die Orchestrierung dieser Web Services in einem Geschäftsprozess mittels WS-BPEL

WS-BPEL unterstützt als Sprache neben einfache Aktivitäten wie “assign”, “invoke”, “throw” auch Konstrukte wie “sequence”, “while”, “repeatUntil” usw. Das macht die Beschreibung von sehr komplexen Geschäftsprozessen möglich, wobei neben der reinen Reihenfolge auch eine komplexere Logik des Prozesses umgesetzt werden kann. Ein Beispiel für Geschäftsprozess ist die Bestellungsabwicklung, die als Beispielprozess in dem DWH von KEIDA abgelegt ist. Die Ausführung des Prozesses ist mit dem Aufruf verschiedener Web

Services verbunden, die Teilaufgaben lösen, z.B. die Abfrage, ob es die bestellten Waren auf Lager gibt, und falls diese nicht da sind, die Bestellung bei den Lieferanten, die Kreditwürdigkeit des Kunde zu überprüfen usw. Somit ist der Prozess für eine vollständige Abwicklung einer Bestellung zuständig. Die einzelnen Aktivitäten, die als WS realisiert sind, werden entweder in dem Unternehmen entwickelt und betrieben oder aber auch von externen Anbietern bezogen.

Das Verteilen der Aufgaben auf einzelnen Web Services macht eine Auslagerung der Funktionalitäten möglich und ist eine der wichtigsten Voraussetzungen bei der Prozessoptimierung. Speziell im Kontext des KEI Frameworks spielen nicht nur die Preise der externen Anbieter für die Nutzung der Webservices eine Rolle, sondern auch von welchem Stromanbieter der Strom zum Betreiben der Rechenzentren bezogen und wie viel Energie verbraucht wird, um die Aktivität auszuführen. Jeder Stromanbieter hat einen eigenen Energiemix und bei der Stromproduktion fallen verschiedene Mengen an CO_2 und Atommüll an. Für die Prozessoptimierung im Fall von KEIDA wird nach dem möglichst geringen CO_2-Ausstoß oder dem produzierten Atommüll gesucht, um den Prozess möglichst ökologisch zu betreiben.

2.2 Green IT

Die Rechner spielen heutzutage eine große Rolle in den meisten Unternehmen. Ein Arbeitsplatz ohne Rechner ist fast undenkbar. Dazu gehören auch die großen Datenzentren, die täglich große Datenmengen bei der Abwicklung der Prozesse in den Unternehmen bearbeiten und speichern. Das Beitreiben dieser Rechenzentren ist mit einem enormen Energieverbrauch verbunden. Dieser steigt ständig und erreicht neue Dimensionen. Der Stromverbrauch in den Rechenzentren weltweit betrug im Jahr 2007 12GW. Vier Jahren später, in 2011, hat sich dieser verdoppelt und stieg auf 24GW an. Für 2013 wird erwartet, dass die verbrauchte Energie in den Rechenzentren weltweit 43GW betragen würde [Va12]. Dieser stetige Anstieg des Energieverbrauchs spiegelt sich nicht nur in den steigenden Kosten, sondern ist mit einer Erhöhung der CO_2-Emissionen verbunden. Diese zwei Faktoren lassen sich durch das Einsetzen neue und zeitgemäße IT Technologien positiv beeinflussen. Diese neuen Technologien sind viel sparsamer und auch leistungsfähiger. Dadurch lässt sich die IT grüner gestalten und an den neuen Anforderungen anpassen.

2.2.1 Green Organisations

Deutschland war schon immer ein Vorreiter in Sachen Nachhaltigkeit. Eine IDC-Umfrage vom Jahr 2008 zeigt, dass schon damals 52% der angefragten deutschen Unternehmen die IT-Infrastruktur grüner gestalten wollten. Laut den Autoren der Umfrage sind die wichtigsten Treiber die steigenden Energiekosten, die sichtbare Erhöhung des „Carbon Footprint“, die große Konkurrenz von Unternehmen außerhalb Europas und der USA, die vergleichbare Leistungen für niedrigere Preise anbieten und auch die EU-Richtlinien für die Effektivität beim Aufbau und Betreiben von großen Datenzentren [MB08].

Die Organisationen sollen aber unter Green IT nicht nur die Auslagerung von Anwendungen auf die Cloud verstehen. Um eine Organisation grün zu gestalten, ist viel mehr verlangt. Energieeffizienz ist nicht nur auf dem Arbeitsplatz und in den Datenzentren erforderlich. Wichtig sind auch energieeffiziente Software und Hardware, sowie energieeffiziente

Produktion und Wiederverwenden der Abfälle. Einige Strategien, wie sich die IT grün gestalten kann, sind bei [Jo12] zu finden:

- *Virtualisierung*. Statt mehrere Server zu betreiben, die nie voll ausgelastet werden, und die meiste Zeit unbenutzt bleiben, besser auf Virtualisierung setzen. Dadurch werden viel weniger Server betrieben, die gut ausgelastet sind, was zu einer Reduktion der Hardware- und Stromkosten führt. Untersuchungen zeigen, dass Windows Server ohne Virtualisierung nur zu 10% ausgelastet werden. Bei Windows Servern mit Virtualisierung steigt die Auslastung auf 60% [CW10].

- *Optimierung des Datacenters und Power Management.* Die Datenzentren sind so zu gestalten/umzubauen, dass diese möglichst wenig Energie verbrauchen. Der größte Teil des Energieverbrauchs fällt auf die Kühlung. Deswegen werden immer neue Ansätze vorgeschlagen, wie sich dieser Verbrauch reduzieren lässt. Ein interessantes Beispiel aus der Perspektive der Green IT ist die Kühlung des Datacenters des Telekommunikationsanbieters BT. Das Rechenzentrum in Frankfurt wird mittels Regenwasser gekühlt [Ra13]. Andere innovative Ansätze sind z.B. unterirdische Eisspeicher oder kaltes Wasser aus Brunnen. Bei der Stromversorgung gibt es neben den Strom aus erneuerbaren Quellen auch ungewöhnliche Ansätze wie z.B. Strom aus der Alge [Ra13]. Bei der Optimierung des Datacenters soll auch berücksichtigt werden, ob sich die Anzahl der Server verringern lässt. Neben der Anzahl sind ihre physische Größe sowie der Energieverbrauch wichtige Eigenschaften. Neue Atom-Server von Dell und HP sollen laut Hersteller zu 90% Platzeinsparrungen gewährleisten [Ra13].

- *Data deduplication*. Die Daten, die im Unternehmen generiert und gespeichert werden, betragen nicht mehr einige Megabytes, sondern sind auf mehrere Terabytes zu beziffern. Das führt zur einen Vergrößerung der Speicherkapazitäten und ist mit Investitionen verbunden. Eine Lösung, wie diese Menge sich verkleinern lässt, stellt die Data deduplication dar. Tests zeigen, dass der Speicherplatz dadurch sehr niedrig gehalten werden kann, ohne Performanceeinbuße. Die Dateien werden komprimiert gehalten, was aber von dem Benutzer unbemerkt bleibt und sein Zugriff auf die Dateien sich nicht ändert. Der Fokus bei dieser Technologie liegt im Sparen der Systemressourcen [Joh12].

- *Cloud Computing*. Immer mehr Anwendungen werden wie Software as a Service in der Cloud angeboten. Die komplexen Aufgaben werden dadurch vom lokalen Rechner auf das Rechenzentrum oder die Cloud ausgelagert. Das macht den Einsatz von s.g. Thin Clients möglich. Die Mitarbeiter im Büro brauchen dann in den meisten Fällen nichts mehr als einen Browser, in dem alle Anwendungen laufen. Der Energieverbrauch kann dadurch enorm gesenkt werden. An dieser Stelle soll auch die Frage nach der Notwendigkeit des Betreibens von eigenen Rechenzentren gestellt werden. Einerseits sind die Datenzentren der Cloud Anbieter viel besser ausgelegt, indem sie viel weniger Strom verbrauchen und auch auf erneuerbaren Energiequellen setzen und andererseits ist der Administrationsaufwand viel kleiner, was zu Personaleinsparungen führen kann.

- *Teleconference/Telecommute/Telework*. Studien [Ve10] zeigen, dass das Benutzen von Teleconference Software in den US-Unternehmen bis zum Jahr 2020 ungefähr 4.6 Millionen Tonnen CO_2 Emissionen sparen kann. Das entspricht 875 000 Autos weniger auf

den Straßen innerhalb eines Jahres und ist mit Ersparnissen von über 19 Milliarden US Dollar verbunden.

- ***Paperless Solutions, Document Management.*** Laut BMU[2] liegt der jährliche Papierverbrauch in Deutschland bei rund 19 Millionen Tonnen und die Hälfte davon fällt auf die Bereiche Presse, Druck und Büromaterial [Bmu11]. Die Herstellung von Papier ist ein energieintensiver Prozess. Für 100 kg Papier werden mindestens 110 kg CO_2 freigesetzt, 5000 Liter Wasser gebraucht und über 1000 Kilowattstunden Energie verbraucht. Heutzutage existieren sehr viele Software-Lösungen, die diesen Verbrauch in den Unternehmen verringern können. Smartphones und Tablets sind längst ein Standard geworden und bieten viele Möglichkeiten, den Papierverbrauch im Büro zu reduzieren. Cloud Dienste wie Evernote[3], Google Docs[4], Dropbox[5] und viele mehr vereinfachen das Erstellen von Notizen und die gemeinsame Arbeit an Office Dokumenten.

2.2.2 Green Technology

In 2.2.1 wurden Strategien gezeigt, wie sich ein Unternehmen an die neuen Anforderungen des nachhaltigen Umgangs mit den knappen Ressourcen mittels IT ausrichten lässt. Dadurch werden hauptsächlich nur CO_2 Emissionen gespart. Es ist aber auch wichtig, dass bei der Produktion, z.B. in der Landwirtschaft, die Chemikalien durch andere Substanzen oder Methoden ersetzt werden, die die Umwelt schonen. Mit diesem Problem beschäftigt sich die Grüne Technologie. Sie ist "technology that has the potential to significantly improve environmental performance relative to other technology" [UN12]. Die Nutzung einer solchen Technologie beinhaltet "the use of environmental technologies for monitoring and assessment, pollution prevention and control, and remediation and restoration" [UN12]. Grüne Technologien lassen sich von der Landwirtschaft über den Gebäudebau oder den Transport von Gütern und Personen bis hin zur Produktion von Lebensmitteln und elektronischen Geräten, die zum Einsatz in jedem Büro kommen, einsetzen. Die Energiegewinnung aus erneuerbaren Quellen wie Wind und Solar ist ein Beispiel für diese grünen Technologien. In dem Kontext der IT beschäftigt sich die Green IT mit dieser Frage. Hier geht es um das Design, die Produktion, das Verschrotten und das Benutzen von Rechnern, Servern und allen dazugehörigen Peripheriegeräten wie Bildschirme, Drucker, Speichermedien, Netzwerkgeräte und Kommunikationssysteme, wobei der Schwerpunkt auf das Minimieren ihres Einflusses auf die Umwelt liegt [Ms08]. So setzt z.B. das Grüne Rechner Design auf neue Materialien und Techniken, ohne das Gerät teurer zu machen oder an Leistung zu sparen. Das beweist auch die Weiterentwicklung der Prozessoren von Single zu Multi-Core. Die neue Multi-Core Chips sind leistungsfähiger und verbrauchen viel weniger Energie. Wie bei der Optimierung der Datenzentren gezeigt wurde, werden die Server auch immer kleiner und dadurch leichter zu kühlen, aber gleichzeitig leistungsfähiger.

2.2.3 Green Processes

Wie gerade gezeigt, unterstützt die IT den ganzen Zyklus vom Design über die Implementierung und das Management bis hin zum Reengineering eines Geschäftsprozesses.

[2] Bundesministerium für Umwelt, Naturschutz und Reaktorsicherheit
[3] www.ervernote.com
[4] www.docs.google.com
[5] www.dropbox.com

Die führenden Größen, nach denen sich diese Schritte orientieren, sind: Zeit, Qualität, Preis und Effektivität. Nachdem die Energieeffizienz und der damit verbundene niedrige CO_2 Ausstoß auf die Tagesordnung stehen, muss beim Gestalten des BPs noch eine Größe berücksichtigt werden, und zwar die Nachhaltigkeit (Abb. 2.2) [NLM11]. Dabei ist es wichtig zu wissen, wie viel Energie jede einzelne Aktivität verbraucht und wie sich dieser Verbrauch verringern lässt. Die Green IT leistet gute Arbeit durch die Bereitstellung von neuen innovativen Softwarewerkzeugen, unter anderem der Analyse, Modellierung und Simulation der Folgen auf die Umwelt dienen und den Energieverbrauch überwachen [Ms08]. Die gesammelte Information kann benutzt werden, um die BPs zu optimieren. Ein solches Werkzeug ist das KEI Framework. Es überwacht, simuliert den Ablauf eines Prozesses und sammelt Informationen über den Energieverbrauch jeder Aktivität. Diese Informationen werden danach aggregiert und gespeichert. Die Analyse dieses Verbrauchs erlaubt die Lokalisierung der größten Stromfresser und eventuell ihre Ersetzung durch energiesparende Lösungen. Dabei ist es wichtig, dass die externen Anbieter Informationen über die Laufzeit und über die gebrauchte und eingesetzte Energie zur Verfügung stellen. Auf dieser Basis kann entschieden werden, ob das Ersetzen der bestehenden Aktivität mit der neuen zu einer Verbesserung der Öko-Bilanz des Prozesses beiträgt.

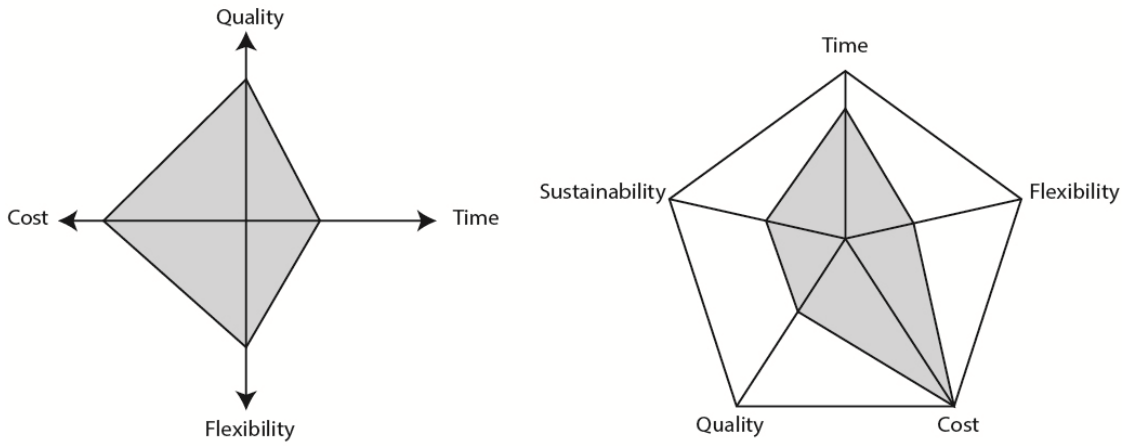

Abbildung 2.2: The devil's pentagon [BS12]

2.3 Business Intelligence Werkzeuge

Business Intelligence (BI)-Lösungen spielen eine große Rolle bei Entscheidungen des Managements in den heutigen Unternehmen. BI Werkzeuge ermöglichen eine umfassende Analyse der täglich entstehenden Daten. Somit sind bestimmte Trends leicht zu erkennen und auf diese Trends kann rechtzeitig und entsprechend reagiert werden. Der gleiche Ansatz wird auch beim KEI Framework verfolgt. Das Framework ist so ausgelegt, dass Informationen über die Laufzeit des BP und seinen Energieverbrauch durch einen ETL-Prozess in einem Data Warehouse gespeichert und diese Informationen grafisch auf einem Dashboard dargestellt werden.

2.3.1 Data Warehouse

Ein DWH wird als ein unternehmensweiter Datenbestand konzipiert. Es steht über den operativen Systemen und speichert alle relevanten Informationen, die an den Interessierten zur Verfügung gestellt werden. Inmon definiert ein Data Warehouse als: "... subject oriented,

integrated, nonvolatile, and time-variant collection of data in support of management's decisions. The data warehouse contains granular corporate data" [In96]. Das Data Warehouse dient als zentraler Speicherplatz für die Informationen, die beim täglichen Geschäft in dem Unternehmen entstehen. Aus der Inmons Definition lassen sich die Haupteigenschaften des DWH ablesen:

- ***themenorientiert***. Die Themen ergeben sich aus den Interessen des Managements. So sind die Entscheidungsträger in der Lage, nach Themen, an denen sie interessiert sind, zu recherchieren [KMU06]. Im Fall von KEI Framework sind diese Themen: Prozesse und die dazugehörigen Prozessaktivitäten.

- ***integriert***. Normalerweise stammen die Daten in einem Data Warehouse aus verschiedenen Quellen. Die Hauptaufgabe bei der Erstellung von einem DWH ist es, die Informationen, die von unterschiedlichen operativen Systeme generiert werden, zu integrieren, sie zu transformieren und unifizieren und so widerspruchsfrei in der Datenbank abzulegen [KMU06].

- ***nicht flüchtig***. Einmal abgelegt in dem Data Warehouse werden die Daten nie gelöscht oder geändert. Das macht eine Analyse der Daten, die in vergangenen Perioden dort abgelegt wurden, möglich [KMU06].

- ***zeitbezogen***. Die Daten, die im DWH abgelegt werden, sind schon aggregiert und repräsentieren einen Zeitraum. Im Unterschied dazu sind die Daten in den operativen Systemen zeitpunktbezogen, d.h. diese werden sofort nach Entstehung erfasst und gespeichert [KMU06].

Die Daten im DWH werden auf Dauer aufbewahrt. Sie dienen der Analyse nicht nur von gängigen Perioden, sondern auch zur Überprüfung wie sich die aktuelle Situation zu den vergangenen Perioden geändert hat. Die Interessierten an dieser Information sind dann in der Lage, sie mit verschiedenen Werkzeugen auszuwerten. Das betrifft auch die Informationen, die das KEI Framework generiert und im DWH speichert. Diese Information spielt eine wichtige Rolle bei Entscheidungen, die die Optimierung des jeweiligen Prozesses betreffen.

2.3.2 ETL-Prozesse

Die Daten, die die operativen Systeme, wie Supply Chain Management, Enterprise Ressource Planning oder E-Procurement liefern, können in verschiedenen Datenformaten oder Strukturen vorliegen. Damit die Daten für die Speicherung in dem DWH vorbereitet werden, sollen diese durch spezielle ETL-Prozesse transformiert werden. Die ETL-Prozesse haben die Aufgabe, diese Daten zu extrahieren, zu transformieren und dann in dem Ziel-Data Warehouse abzuspeichern.

Als erster Schritt wird die Extraktion angesetzt. Ihre Aufgabe ist es, die Daten aus den operativen Systemen zu holen. Da die Abspeicherung der Daten in den operativen Systemen wegen der Größe der generierten Daten von sehr kurzer Dauer ist, soll diese Extraktion periodisch erfolgen. Andere denkbaren Szenarien sind, dass sie auf Anforderung erfolgt oder ereignisgesteuert ist.

In dem zweiten Schritt liegt eine Transformation der gewonnenen Daten. Sie werden in betriebswirtschaftlich interpretierbare Daten umgewandelt. Dieser Schritt besteht aus vier Teilschritten: Filterung, Harmonisierung, Aggregation und Einreichung [KMU06].

Nachdem die Daten extrahiert und transformiert wurden, sind diese bereit, im DWH abgelegt zu werden. Dies erfolgt in einem dritten Schritt, der als Load bezeichnet wird. Die so abgelegten Daten sind dann bereit, durchsucht oder analysiert zu werden.

2.3.3 Dashboard

Die Daten, die das DWH aufbewahrt, sind in den meisten Fällen Zahlen. Würden diese in einer Tabelle auf dem Bildschirm angezeigt, ist für den Analysten sehr schwer, nach Abhängigkeiten zu suchen. Dafür sollen die Daten visualisiert und in einer Form bereitgestellt werden, die für jeden verständlich ist. Die Menschen sind in der Lage, sehr schnell Zusammenhänge in grafisch dargestellten Zahlenmengen zu finden, was beim Anschauen einer Excel-Tabelle mit der gleichen Information fast unmöglich erscheint. Dazu wird ein Dashboard eingesetzt. Ein Dashboard ist "a visual display of the most important information needed to achieve one or more objectives; consolidated and arranged on a single screen so the information can be monitored at a glance" [Fe04].

Die Dashboards stellen die wichtigste Information dar und ermöglichen einen schnellen Überblick über die wichtigsten Entwicklungen. Die Charakteristiken eines Dashboards sind wie folgt [RC04]:

- Auf dem Bildschirm sind verschiedene Metriken auf einem einzigen Screen gezeigt.
- Die angezeigte Information ist auf dem höchsten Niveau der Granularität.
- Intuitive Indikatoren zeigen die wichtigsten Performance Indikatoren (KPI).
- Auf dem Dashboard werden nur die Folgen und nicht die Ursachen angezeigt.
- Der Aufbau ist intuitiv und einfach und kann leicht verstanden und bedient werden.
- Die angezeigte Information wird automatisch aktualisiert.

Das KEI Framework beschäftigt sich mit der Optimierung der Prozesse und damit, wie sich diese dadurch nachhaltiger betreiben lassen. Deswegen wird hier nicht über KPI geredet sondern über Key Ecological Indicators (KEI). Die wichtigste Größe in diesem Fall ist der Energieverbrauch. Aus dem Energieverbrauch und das vorliegende Energiemix des Stromzulieferers lassen sich auch weitere KEIs berechnen. Das sind die Grüne Energie, die CO_2 Emissionen sowie der radioaktive Müll, die bei der Produktion der benötigten Energie als Endprodukt anfallen und natürlich nicht an letzter Stelle auch die Kosten, die beim Ausführen des Prozesses entstehen.

3 Konzept

3.1 Zielbestimmung

Die Nachhaltigkeit ist zu einem zentralen Thema in den heutigen Unternehmen geworden. Die Energieeffizienz in jedem Bereich des Unternehmens spielt eine wesentliche Rolle. Dabei geht es nicht nur um Energieeffizienz der Hardware, sondern auch der Software. Von der Software wird erwartet, dass diese auch zu einem sparsamen Ablauf der Prozesse beiträgt. Deswegen wird bei der Optimierung der Prozesse, wie in 2.2.1 gezeigt wurde, neben Größen wie Zeit, Qualität, Preis und Effektivität auch die Größe Nachhaltigkeit berücksichtigt. Es wird Wert darauf gelegt, wie viel Strom der Prozess bei einem Durchlauf verbraucht und wie viel CO_2 bei der Produktion dieses Stroms in die Atmosphäre ausgestoßen wird. Das KEI Framework stellt einen solchen Einsatz dar und versucht, Antworten auf diese Fragen zu geben. Dadurch ist eine Optimierung der Prozesse im Einklang mit den neuen Anforderungen möglich.

Auf Basis des KEI Frameworks existieren zwei Teilprojekte, die es unterstützen: KEI Dashboard und Greevi App. Das erste ist eine Dashboard Implementierung und wurde nur mit dem Ziel geschaffen, zu zeigen, dass die Information, die im DWH gespeichert wird, sich leicht in der gewünschten Form visualisieren lässt. Das Greevi App dagegen unterstützt das Visualisierungswebservice “ViproWS” und ermöglicht über ein Web-Interface auf Basis der Daten im DWH sowie verschiedene Templates eine Visualisierung des Prozesses. Somit werden die Problemzonen direkt auf dem Prozessmodell graphisch angezeigt. Unter Problemzonen sind die Aktivitäten zu verstehen, die z.B. zu viel Energie verbrauchen oder bei denen im Energiemix des Stromanbieters zu viel Energie aus Atomkraftwerken benutzt wird. Dadurch sinkt der Teil an grüner Energie und diese Aktivität wird entsprechend dem vorgegebenen Mapping auf dem Bild gefärbt oder skaliert.

Die zwei Teilprojekte sind mit verschiedenen Technologien realisiert. Das KEI Dashboard setzt direkt auf JSP (Java Server Pages), die Greevi App hingegen auf das Vaadin Framework. Ziel dieser Studienarbeit ist es, diese zwei Prototypen zu vereinigen, zu erweitern, damit eine vollständige Analyse von jedem im DWH abgelegten Prozess möglich ist. Es wird nach einer Lösung gesucht, die die benötigte Funktionalität und Information an einer Stelle anbietet und die einfach zu bedienen ist.

3.2 Funktionale Anforderungen

Aufbereitung historischer Informationen: Die Daten im DWH erfassen jeden Durchlauf eines Prozesses und zeichnen auf, wann dieser gestartet und beendet wurde. Dabei werden alle einzelnen Aktivitäten, die im Prozess aufgerufen werden, deren Start- und Endzeitpunkte, die Energie, die sie verbraucht haben, und die dazugehörigen KEIs (CO_2-Ausstoß, Atommüll, Prozent grüner Energie), berücksichtigt und abgelegt. Diese Granularität ist jedoch zu hoch und bevor die Daten auf dem Dashboard visualisiert werden, sollen diese aggregiert werden. Deswegen soll das System eine Aggregation der Daten nach Stunde, Tag, Woche und Monat bieten. Dabei ist es nicht wichtig, wie oft der Prozess und die dazugehörige Aktivitäten aufgerufen wurden, sondern ihr Energieverbrauch, der Prozent Grüner Energie, der produzierte CO_2, die Atommüllmengen und die verursachten Kosten.

Darstellung historischer Informationen: Wie in 2.3.3 deutlich wurde, trägt eine graphische Darstellung von Zahlenreihen dazu bei, dass die Zusammenhänge, die dahinter stecken, von den Menschen leichter analysiert werden können. Das System soll deswegen die aufbereiteten historischen Informationen in einer geeigneten graphischen Form dem Benutzer bereitstellen. Als graphische Darstellungen werden verschiedene Graphiken vorausgesetzt, wie Balken- und Liniendiagramme oder Tachometer.

Visualisierung eines Prozesses: Bei der Analyse eines Prozesses ist neben der graphischen Darstellung der historischen Informationen eine weitere Analyse des Prozesses mit Hilfe von ViproWS möglich. Das System soll deswegen die Möglichkeit bieten, dieses Webservice zu benutzen und eine geeignete Visualisierung der Struktur des Prozesses zu ermöglichen. Der Benutzer soll an erster Stelle in der Lage sein, einen bestimmten Prozess und die Instanzen aus einem Zeitraum, in dem dieser gelaufen ist, auszuwählen. Der Benutzer soll weiter die Möglichkeit haben, Funktionen und dazugehörige Funktionswerte festzulegen, um bestimmte Aspekte der generierten Daten der ausgewählten Prozessinstanzen untersuchen zu können. Wichtige Aspekte dabei sind, ob der Mittelwert kleiner als vom Benutzer eingegebenen Funktionswert ist, ob das Maximum über alle Instanzen kleiner oder größer ist usw. Danach soll der Benutzer die Wahl zwischen verschiedenen vorgegebenen Mappings treffen können, um eine geeignete Visualisierung des Prozesses zu erhalten.

Prognosefunktionalitäten durch Annotationen von Aktivitäten: Die Visualisierung der Prozesse erfolgt anhand der im DWH gespeicherten Daten. Diese werden benutzt, um die Werte, die an das ViproWS geschickt werden, zu berechnen. Wie in 2.1. gezeigt wurde, könnten die einzelnen Aktivitäten eines Prozesses auch von externen Anbietern bezogen werden. Die Unternehmen, die ihre Prozesse mit dem KEI Framework optimieren wollen, werden dann von den WSs Anbieter erwarten, dass diese nicht nur die Preise, sondern auch Informationen über den Energieverbrauch, CO_2-Ausstoß usw. angeben. Das System soll dann dem Benutzer die Möglichkeit bieten, diese neue Information mit der Information, die im DWH steht, zu ersetzen. Dadurch ist der Benutzer des Systems in der Lage, eine neue temporäre Visualisierung des Prozesses zu generieren, die auf Basis der neuen Daten gemacht wurde.

Vergleich von zwei Darstellungen eines Prozesses: Die vorherigen zwei Anforderungen beschreiben, dass der Benutzer des Systems die Möglichkeit haben soll, zwei Visualisierungen zu bekommen, einmal mit den originalen Daten und einmal mit den geänderten Daten. Damit dieser die Unterschiede leicht erkennen kann, soll er in der Lage sein, die zwei Abbildungen zu vergleichen. Daraus erfolgt die Anforderung an das System, dass dieses dem Benutzer die Möglichkeit geben soll, die von dem ViproWS generierten Bilder nebeneinander zu stellen, um sie vergleichen zu können.

3.3 Nicht-funktionale Anforderungen

Technische Anforderungen:

- Als Programmiersprache wird Java vorausgesetzt. Die Java Version soll 6 oder höher sein.
- Die Architektur soll als Client/Server konzipiert werden.

- Das Client soll webbasiert und in allen gängigen Webbrowser abrufbar sein.
- Der Server soll Apache Tomcat 6 oder eine höhere Version sein.

Benutzerfreundlichkeit: Die Benutzerfreundlichkeit der GUI ist einer der wichtigsten Aspekte, die bei der Akzeptanz seitens der Benutzer eine Rolle spielen.

- Die graphische Oberfläche soll zeitgemäß gestaltet werden. Diese soll übersichtlich und aufgeräumt sein und alle benötigten Funktionalitäten anbieten. Die Benutzer sollen ohne großen Aufwand den Umgang mit dem System lernen.
- Die Funktionen, die bereit stehen, sollen leicht bedienbar sein. Bei falschen Eingaben sollen entsprechende Fehlermeldungen angezeigt werden.

Zuverlässigkeit: Das System unterstützt den Entscheidungsträger bei der Optimierung eines Prozesses. Dabei spielt die Richtigkeit der angezeigten Werte eine große Rolle. Aus diesem Grund soll das System immer die geforderten Informationen korrekt berechnen.

4 Design

In Kapitel 3 ist unter Technische Anforderungen Java als Programmiersprache zu sehen. Diese Voraussetzung führte dazu, dass beim Design des KEIDA und bei der nachfolgenden Implementierung die Grundprinzipien der Java EE Architektur befolgt wurden. In 4.1 werden die Java EE Architektur sowie das Model-View-Controller (MVC) Entwurfsmuster kurz erläutert. Anschließend werden ein Überblick über das KEIDA Design und eine Beschreibung der KEIDA Architektur gemacht.

4.1 Java EE Architektur

Java EE ist eine Plattform zur Entwicklung von unternehmensweiten Anwendungen [IT13]. Java EE ist auf Basis des 4-Tier-Architekturmodells aufgebaut – Client-, Web-, EJB[6]- und EIS[7]-Tier. Java EE ist komponentenbasiert und ermöglicht den Aufbau von verteilten Anwendungen. Die Komponenten, die eine Anwendung aufbauen, laufen auf verschiedenen Schichten der Architektur, sie sind lose, gekoppelt und unabhängig voneinander. Das führt zu einer klaren Trennung nicht nur der Funktionalitäten und Zuständigkeiten der Komponenten, sondern auch zu einer engen Spezialisierung seitens der Entwickler.

KEIDA setzt auf die Java EE Plattform. Als Client dient ein Web-Browser. Die Web- und EJB-Tier sind in einem Apache Tomcat Server untergebracht. Eine relationale Datenbank (MySQL Server) sowie das ViproWS, das auf einen Apache Tomcat Server läuft, sind als EIS-Tier eingesetzt.

Mit Java EE lassen sich Webanwendungen erstellen. Bei der Entwicklung der Webanwendungen hat sich die Model-View-Controller (MVC) Architektur bewiesen. Heutzutage sind viele Open Source und kommerzielle Frameworks auf ihrer Basis aufgebaut. Die Architektur teilt, wie auch in der GUI-Programmierung der Fall ist, die Anwendung in drei Schichten. Das sind die Modell-, Präsentations- und Controllerschicht. Diese drei Schichten übernehmen spezifische Aufgaben und haben verschiedene Verantwortlichkeiten gegenüber den anderen Schichten [SSJ02]. Die Aufgaben werden dann in verschiedenen Kategorien von Objekten definiert: “the objects that deal with presentation aspects of the application, objects that deal with the business rules and data, and objects that accept and interpret user requests and control the business objects to fulfill these request” [SSJ02]. Die MVC-Architektur erlaubt z.B. das Ersetzen der Präsentation, ohne dass dabei das Modell geändert wird.

Model-View-Controller Architektur. Die MVC-Architektur führt zu einer klaren Trennung der Aufgaben. In einer GUI sowie in einer Webanwendung interagiert der Benutzer mit der Anwendung und erwartet eine entsprechende Ausgabe. In einer GUI-Anwendung gibt der Benutzer eine Eingabe über die graphische Oberfläche, die Anwendung nimmt dann diese Eingabe, bearbeitet sie und gibt eine Antwort zurück, indem sich die GUI neuzeichnet. Im Fall einer Web Anwendung ist die GUI in den meisten Fällen in einem Webbrowser dargestellt. Der Benutzer stellt über diesen Browser die Anfragen an den Web Server. Der

[6] Enterprise Java Bean
[7] Enterprise Information System

Server bearbeitet die Anfragen, wählt die Präsentation (HTML, PDF, SVG usw.) und schickt sie zurück an den Benutzer.

Die drei Teile der MVC-Architektur sind auf Abb. 4.1 zu sehen. Diese sind:

- Model: Das Model implementiert die Geschäftslogik. Das Model hat Zugriff auf die persistenten Daten und hat die Funktionalitäten, diese zu bearbeiten und sie in einer geeigneten Form dem View bereitzustellen.

- View: Das View ist zuständig für die Repräsentation der Daten, die das Model zurückgibt. Das View kann von dem Model neue Daten erfordern und diese in einer geeigneten Form darstellen. Über das View werden die Befehle an den Controller übergeben.

- Controller: Der Controller ist für den gesamten Flow der Anwendung zuständig. Anhand der Eingaben des Benutzers wird eine entsprechende Funktion ausgeführt. Der Benutzer schickt Eingaben in Form von HTTP GET oder POST Anfragen. Anhand dieser Anfragen entscheidet sich der Controller, welche Model und View gebraucht werden, um die Information zu bearbeiten und in einer geeigneten Form zu präsentieren.

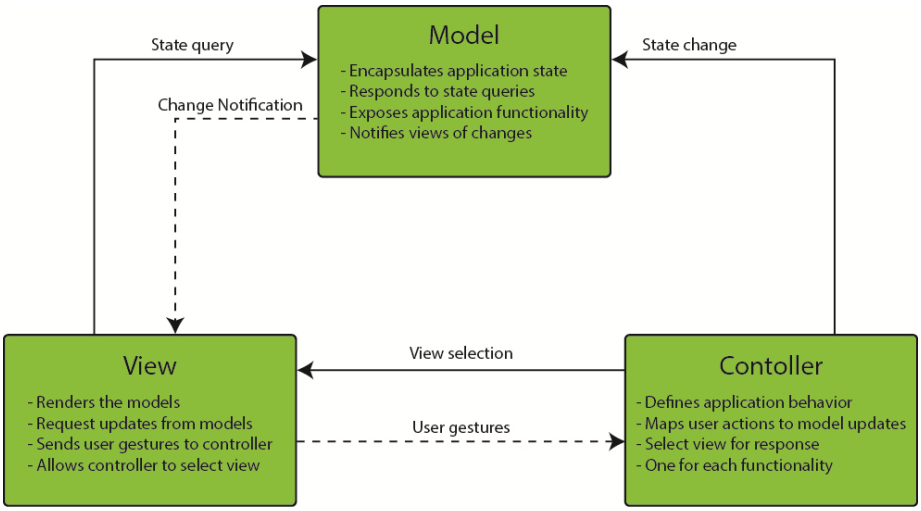

Abbildung 4.1: MVC Entwurfsmuster [NB13]

In einer Webanwendung wird der Controller in der Regel als ein Servlet implementiert. Der Servlet hat die Verantwortung, anhand der Eingabe die passende Aktion zu unternehmen. Für das View könnten Java Server Pages (JSP) oder Java Server Faces (JSF) eingesetzt werden. Das ist aber nicht zwingend, weil die Antwort auch in einer anderen Form zurückgegeben werden kann, z.B. in Form einer pdf-Datei. Die Daten in der Datenbank werden über Session Beans und Data Acess Objects (DAO) aufgerufen und bearbeitet. Diese bilden das Model der Webanwendung. (Abb. 4.2)

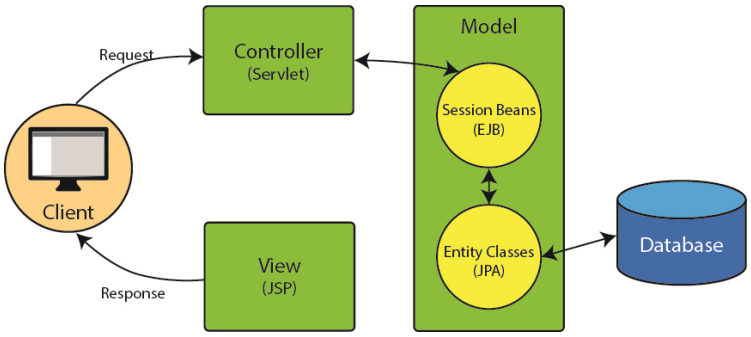

Abbildung 4.2: Architektur einer Webanwendung [NB13]

In der MVC-Architektur finden viele andere Entwurfsmuster wie Composite View, Front Controller, View Helper, Data Access Object ihren Platz. Eine volle Liste sowie ausführliche Beschreibungen der einzelnen Muster findet man in J2EE Core Patterns [Al03].

4.2 KEIDA – Überblick

KEIDA (KEI Dashboard) ist eine typische Webanwendung. Sie setzt auf die dreischichtige Architektur - Client, Web und EIS. Der Benutzer kann auf KEIDA über einen Browser zugreifen. Die Browser-Anfragen werden von einem Tomcat Server, auf dem KEIDA läuft, abgefangen und bearbeitet. Die Daten, auf die das System zugreift, befinden sich auf einem MySQL-Server. KEIDA greift auf das ViproWS auf, um ein BP zu visualisieren. Die Anwendung besteht aus zwei Hauptteilen - Dashboard und Visualisierungswerkzeug.

Dashboard. Das Dashboard ist nach den Anforderungen des Systems, die in Kapitel 3 beschrieben wurden, aufgebaut. Es ist so konzipiert, dass der Benutzer eine Periode auswählen kann und die entsprechende Information entweder alle Prozesse oder nur einen einzelnen Prozess sehen und analysieren kann. Es ist auch möglich, dass der Benutzer auch nach einem bestimmten Prozess suchen kann.

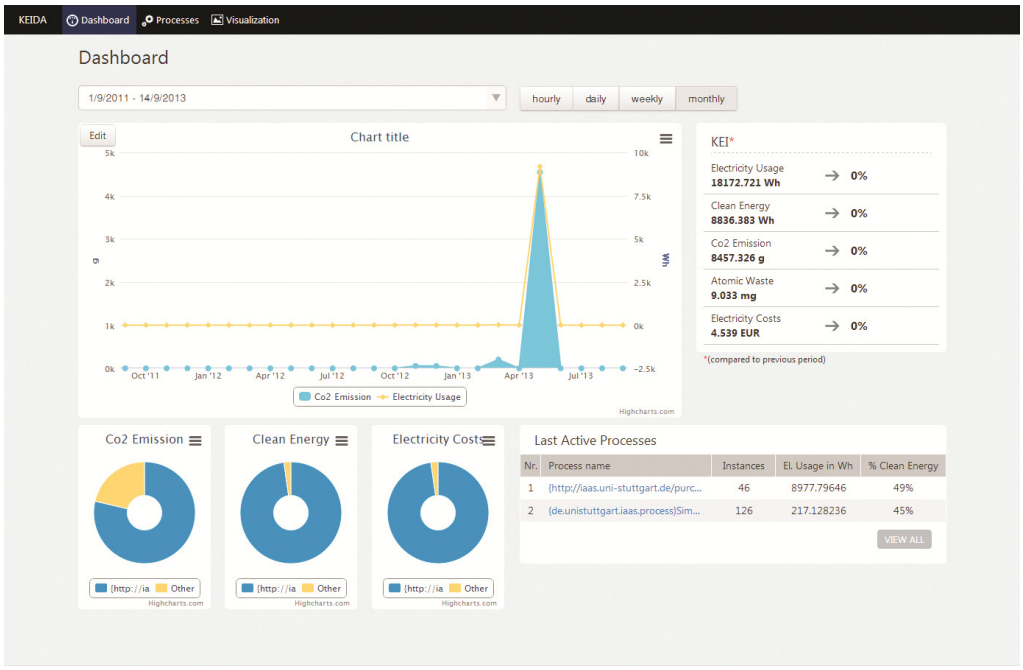

Abbildung 4.3: KEIDA Dashboard

Visualisierungswerkzeug. Der Benutzer kann mittels eines Konfigurationsassistenten die von ViproWS benötigte Information angeben, um einen Prozess zu visualisieren. Weiter ist es auch möglich, dass die Daten, die das System zu jeder Aktivität des Prozesses berechnet, von dem Benutzer geändert werden. Dadurch bekommt der Benutzer eine zweite optimierte Visualisierung des Prozesses. Das Visualisierungswerkzeug bietet die Möglichkeit, die zwei graphischen Repräsentationen nebeneinander darzustellen und ermöglicht somit einen Vergleich beider Darstellungen.

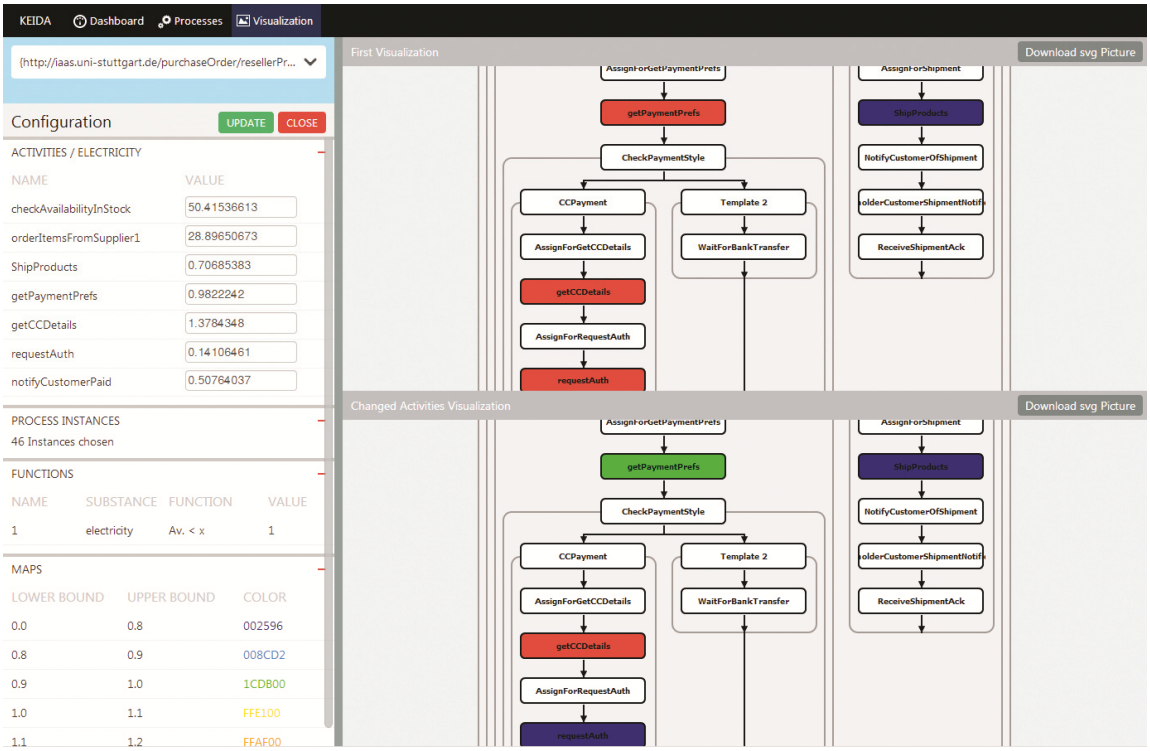

Abbildung 4.4: KEIDA Visualisierungswerkzeug

4.3 KEIDA – Design

Beim Studieren der Anforderungen an das System und der Funktionalitäten, die es haben soll, wurde sofort klar, dass eine Teilung in Dashboard und Visualisierungswerkzeug unbedingt nötig ist. Die zwei Teile beziehen sich auf die gleichen Daten, haben jedoch zwei völlig unterschiedliche Funktionen. Die Anforderungen an das Dashboard setzen eine graphische Darstellung der historischen Daten voraus. Mit dem Visualisierungswerkzeug sollte eine graphische Darstellung der Struktur des BPs mittels ViproWS möglich sein.

Das Dashboard ermöglicht die Auswahl einer Periode, für die die Information über alle gelaufenen Prozesse angezeigt wird sowie die Suche nach einem Prozess. Dadurch kann der Benutzer nicht nur alle Prozesse, sondern auch einzelne Prozesse analysieren. Für das Visualisierungswerkzeug ist eine Art Konfigurationsassistent erforderlich. Mit seiner Hilfe soll der Benutzer in der Lage sein, verschiedene Parameter einzustellen, die für eine Visualisierung benötigt werden. Der Benutzer soll auch die Möglichkeit haben, die berechneten Werte, die den Prozess-Aktivitäten zugewiesen sind, zu ändern und eine zweite Visualisierung vom System anzufordern. Die wichtigste Aufgabe dabei ist es, diese zweite Visualisierung mit der ersten vergleichen zu können.

4.4 KEIDA – Architektur

KEIDA ist als eine Webanwendung realisiert und über einen Browser zugänglich (Abb. 4.5). Diese Anwendung läuft auf einem Web Server (Apache Tomcat). Sie setzt auf die MVC-Architektur und ist in drei Schichten aufgeteilt, die in 4.4.1 näher erläutert werden. Die Daten, die auf dem Dashboard graphisch angezeigt werden, befinden sich auf einem Database Server (MySQL) und KEIDA greift auf sie über eine jdbc-Verbindung zu. Bei der Visualisierung der Processen wird das Vipro Webservice gebraucht, das in einem anderen Web Server (Apache Tomcat) untergebracht ist. Die folgenden zwei Unterkapitel geben einen tieferen Überblick der serverseitigen sowie clientseitigen Architektur.

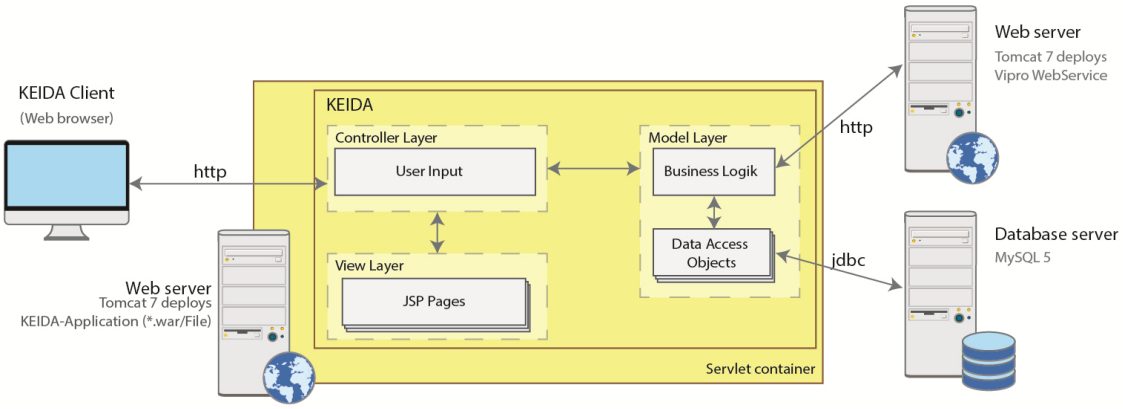

Abbildung 4.5: KEIDA Architektur

4.4.1 Serverseitige Architektur

Die serverseitige Architektur ist entsprechend der Darstellung auf Abb. 4.5 aufgebaut. Sie setzt auf die MVC-Architektur, die eine klare Trennung der einzelnen Schichten und deren Zuständigkeiten liefert. Diese Tatsache veranschaulicht auch das verallgemeinerte Sequenzdiagramm auf Abbildung 4.6. In der MVC-Architektur dient ein Servlet als Front Controller und übergibt die Aufgaben an den unter ihm stehenden Controller. Diese Controller sind eng spezialisiert und wie in Kapitel 5 deutlich wird, nur für bestimmte Aufgaben zuständig. Diese Controller kennen die Modelle, die die benötigte Information liefern können und leiten die von dem Request übergebenen Parameter an diese Modelle weiter. Die von dem Modell zurückgegebene Information wird zusammen mit dem Namen des Views an den Front Controller zurückgeliefert. Dieser bereitet die Antwort vor und schickt sie an den Browser. Das Model von KEIDA ist auf verschiedenen Services aufgebaut. Diese Services sind unabhängig voneinander und sind in der Lage, direkt auf die Daten in der Datenbank zuzugreifen oder auch auf das ViproWS. Der Front Controller nimmt wie oben beschreiben diese Daten von dem Modell und fügt sie in die passende Präsentation ein, die dann an den Browser geschickt wird.

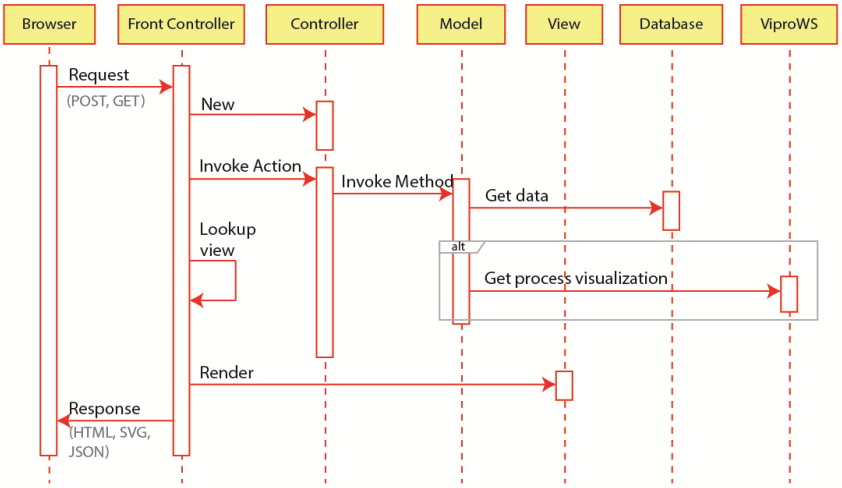

Abbildung 4.6: Bearbeitung einer Anfrage in KEIDA

Präsentationsschicht (View Layer): Die Präsentationsschicht ist hauptsächlich auf JSP Dateien aufgebaut. Hier wird auf dem Composite View Pattern gesetzt, weil mehrere statische Elemente einer Web Seite sich immer wieder wiederholen und diese sich dadurch nur einmal definieren und in mehrere Seiten oder Views einfügen lassen. Auf Basis der Anforderungen an das System wurden die folgenden Ansichten definiert: Dashboard für alle Prozesse, Dashboard für einzelne Prozesse, Suche, Konfigurationsassistent, Prozessvisualisierung und Vergleich zweier Prozessvisualisierungen.

Modelschicht (Model Layer): Die Modelschicht enthält die Geschäftslogik des Systems. Sie ist in einzelnen Beans verteilt. Diese haben Zugriff auf die Businessobjekte und sind so aufgeteilt, dass sie jede spezifische Funktionalität bereitstellen. Diese Gruppierung der Funktionalitäten macht eine zukünftige Erweiterung leichter, weil eine definierte Grenze zwischen den Zuständigkeiten existiert. Die Trennung nach Funktionalitäten führt zu einem überschaubaren Code, der leicht zugeordnet werden kann. Das Model ist auf einzelnen getrennten Komponenten aufgebaut. Dadurch ist leicht nachvollziehbar, welche Komponente welche Zuständigkeiten hat. Diese Aufteilung ermöglicht eine Wiederverwendung von Funktionalitäten an verschiedenen Stellen.

Controllerschicht (Controller Layer): Die Controllerschicht setzt auf das Front Controller Entwurfsmuster. Der Controller steuert die Anwendung, indem dieser die Anfragen des Benutzers bearbeitet und an das richtige Model weiterleitet. Der Controller ist in Kombination mit Command Entwurfsmuster aufgebaut. Die HTTP Anfragen haben eine spezifische Struktur, anhand derer sie einem Model zugewiesen werden. Dieses Model hat die passende Businesslogik und liefert die angefragte Information. Die Anwendung kann dadurch das richtige View auswählen, um die passende Präsentation der Information zu gewährleisten.

4.4.2 Clientseitige Architektur

Der Benutzer der Anwendung kann auf sie über einen Browser zugreifen. Wie schon erläutert, besteht KEIDA aus zwei Hauptteilen, nämlich Dashboard und Konfigurationsassistent. Da das Dashboard Informationen graphisch darstellt, wird auch keine große Interaktion mit dem Benutzer vorausgesetzt. Dagegen ist der Konfigurationsassistent komplexer und begleitet den

Benutzer beim Visualisierungsprozess. Die GUI, die hier aufgebaut ist, setzt wie jede andere auch auf MVC. Das Muster ist ein Bestandteil einer GUI-Anwendung und erleichtert den Aufbau des Konfigurationsassistenten. Hier hat das Model im Unterschied zu den MVC auf dem Server einen Einfluss auf die Präsentation. Auf dem Server wird das Model so konzipiert, dass es auch durch ein anderes ersetzt werden kann, ohne dass etwas in dem View oder Controller geändert werden muss. In der GUI hat jedoch das Model eine Referenz zum View und benachrichtigt es wann es sich updaten soll. Das Modell enthält alle Informationen, die bei der Einstellung einer Visualisierung eingegeben werden, und sorgt dafür, dass das View rechtzeitig auf geänderte Informationen entsprechend reagiert. Das View leitet die neu eingefügten Informationen an den Controller weiter. Dieser benachrichtigt seinerseits das Modell. Darauf bauen alle graphischen Elemente auf, die eine Interaktion mit dem Benutzer ermöglichen.

5 Implementierung

Im folgenden Kapitel wird eine ausführliche Beschreibung der Implementierung des Konzepts, das in Kapitel 4 eingeführt wurde, gemacht. Das Kapitel fängt mit einer kurzen Einführung der eingesetzten Frameworks an: SpringMVC auf der Serverseite und jQuery auf der Clientseite. Danach folgen Beschreibungen der konkreten Implementierung der Konfiguration sowie der View-, Model- und Controller-Schichten auf dem Server. Der Kapitel schließt mit einer Übersicht der clientseitigen Implementierung ab.

5.1 SpringMVC und jQuery

Die Serverseitige Implementierung setzt, wie schon in Kapitel 4 gezeigt wurde, auf die Java EE Technologie auf. Diese bietet die Grundfunktionalitäten, die den Aufbau einer Webanwendung ermöglichen. Dieser Aufbau ist bei jedem Projekt mit der Implementierung von spezifischen Funktionalitäten, die immer wieder vorkommen und sich als Best Practices bewiesen haben, verbunden. Das ist z.B. die MVC-Architektur bei einer Webanwendung. Diese Funktionalitäten werden in den Frameworks implementiert und dem Entwickler zur Verfügung gestellt. Ein solches Framework ist SpringMVC, das zum Einsatz beim Aufbau der serverseitigen Funktionalitäten von KEIDA kommt.

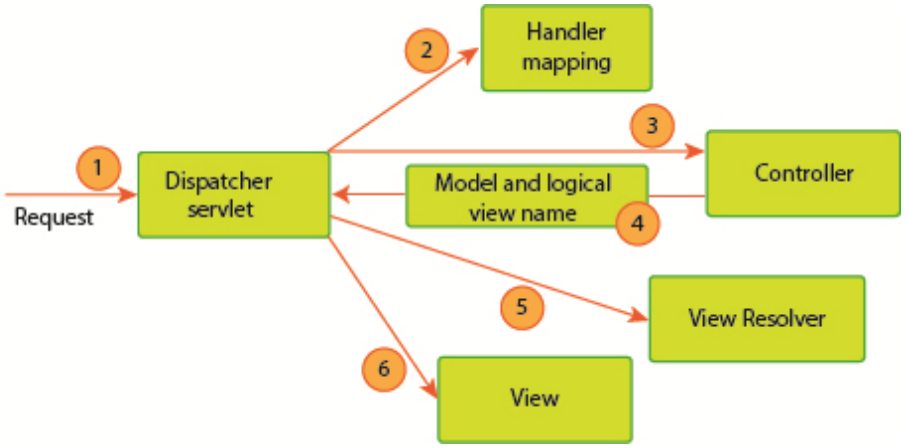

Abbildung 5.1: Dispatcher servlet workflow [Wa11]

Spring MVC [Wa11] ist um den *DispatcherServlet* aufgebaut. Dieser setzt auf das Front-Controller Entwurfsmuster. Der Front-Controller ist der Eingangspunkt für alle Anfragen, die der Benutzer an die Webanwendung macht. Die Aufgabe der *DispatcherServlet* ist es, diese Anfragen an den zuständigen Controller weiterzuleiten. Die vom Entwickler geschriebenen Controller sind mit *@Controller* annotiert. In den Controller-Klassen sind die Methoden mit *@RequestMapping* zusätzlich annotiert und für spezifische Aufgaben zuständig. Diese Informationen werden in dem zweiten Schritt "Handler mapping" im Request-Workflow benötigt (Abb. 5.1, Schritt 2). Somit ist der *DispatcherServlet* in der Lage, den richtigen Controller auszuwählen (Abb. 5.1, Schritt 3). Der Controller seinerseits hat eine Referenz zum Model und leitet ihm die Daten, die der Benutzer geschickt hat (Abb. 5.1, Schritt 4). Das Model implementiert die Geschäftslogik, die anhand der Anfrage des Benutzers und der mitgeschickten Daten eine entsprechende Antwort liefern kann. Der Controller gibt an den *DispatcherServlet* nicht nur die Daten zurück, die das Model geliefert hat, sondern auch den

Namen des Views, das bei der Präsentation dieser Daten benutzt werden soll. Dieses View wird von dem *ViewResolver* bestimmt (Abb. 5.1, Schritt 5). Der *ServletDispatcher* kann jetzt dieses View nehmen (Abb. 5.1, Schritt 6), um die Antwort, die an den Benutzer zurückgeschickt wird, richtig zu formatieren. In den meisten Fällen wird als View eine HTML-Template-Technologie eingesetzt z.B. JSP oder Tiles. Es ist aber auch möglich, einfaches XML oder JSON an den Browser zurückzuschicken.

Auf die Clientseite werden bei der GUI-Programmierung JavaScript und HTML benutzt. Als Framework wird hier jQuery eingesetzt. jQuery implementiert wie Spring MVC wichtige, sich immer wieder wiederholende Funktionalitäten und stellt diese in einer einfachen und leicht verständlichen Form an den Front-End Entwickler bereit. Einige davon sind:

- Document Object Model (DOM) und Cascading Style Sheet (CSS) Manipulation
- Event-System
- AJAX Funktionalitäten
- Möglichkeit, die Bibliothek mit Hilfe von benutzerdefinierten Plug-Ins um eigene Funktionalitäten zu erweitern
- Animationen von HTML-Elementen.

Bei der Visualisierung der Graphiken auf dem Dashboard wird die Highcharts JavaScript Bibliothek benutzt. Diese generiert anhand von JSON Objekte die graphische Darstellung der Zahlenreihen, die auf dem Server berechnet wurden.

5.2 Konfiguration

Der Eingangspunkt jeder Spring MVC Anwendung ist, wie in 5.1 gezeigt wurde, der *ServletDispatcher*. Damit dieser als solcher erkannt wird, wird er in der web.xml Datei definiert:

```
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
   <servlet-name>spring</servlet-name>
   <url-pattern>/</url-pattern>
</servlet-mapping>
```

Abbildung 5.2: Servlet- und Servlet-Mapping-Definitionen in web.xml

Die Servlet-Definition besagt, dass der *DispacherServlet* als erster geladen werden soll und die *Servlet-Mapping* zeigt, dass alle Anfragen an diesen weitergeleitet werden sollen. *Servlet-Name* ist der Name der Spring-Konfigurationsdatei. Im Fall von KEIDA ist das die *spring-servlet.xml*. Die Konfiguration von KEIDA wurde auf mehrere xml-Dateien verteilt. Die einzelnen Dateien enthalten spezifische Teile dieser Konfiguration (Abb. 5.3). Sie erfolgt

mittels Java Beans, die zur Initialisierungszeit in den einzelnen Klassen durch Dependencies Injection geladen werden. So ist z.B. in der keida-db.xml-Datei der “dataSource”-Bean, der eine Verbindung mit der Datenbank aufbaut, definiert (Abb. 5.6). Dieser braucht neben den Namen der Datenbank den Pfad, den Benutzernamen und das Passwort. Dadurch kann Spring zur Initialisierungszeit eine Verbindung mit der Datenbank aufbauen. In dieser Datei sind alle anderen Beans, die einen Zugriff auf die Datenbank benötigen, definiert.

```
<!-- MySQL Data Base configuration -->
<import resource="keida-db.xml" />
<!-- Services Beans -->
<import resource="keida-services.xml" />
<!-- ViproWS Configuration -->
<import resource="keida-viproWS-configuration.xml" />
<!-- Main Dashboard Configuration -->
<import resource="keida-homepage-configuration.xml" />
<!-- Process page Configuration -->
<import resource="keida-processPage-configuration.xml"  />
<!-- Visualization page Configuration -->
<import resource="keida-visualization-mapping.xml"  />
```

Abbildung 5.3: Importieren von Konfugurationsdateien in spring-servlet.xml

In der *spring-servlet.xml* ist angegeben, wo sich die Controller befinden und dass diese durch Annotation gekennzeichnet sind:

```
<context:component-scan base-package="de.unistuttgart.iaas.keida.controller" />
<mvc:annotation-driven/>
```

Abbildung 5.4: Auszug aus der spring-servlet.xml -Datei

Weiter folgt die Definition des *ViewResolvers*. Als Template-System wird in KEIDA auf Tiles 2.2 gesetzt. Der *Tiles2*-Bean definiert, in welchem Ordner sich die Template-Dateien befinden sowie dass das *TilesViewResolver* gebraucht wird, um den entsprechenden View zu finden:

```
...
<bean class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
        <property name="definitions">
            <list>
                <value>/WEB-INF/views/**/views.xml</value>
            </list>
        </property>
    </bean>
<bean class="org.springframework.web.servlet.view.tiles2.TilesViewResolver" />
...
```

Abbildung 5.5: Tiles 2.2 Konfiguration

```
...
<bean id="dataSource" class="org.springframework.jdbc.datasource…">
      <property name="driverClassName" value="com.mysql.jdbc.Driver" />
      <property name="url" value="jdbc:mysql://localhost:3306/kei_dwupdate" />
      <property name="username" value="root" />
      <property name="password" value="" />
</bean>
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource"> <ref bean="dataSource"/> </property>
</bean>
...
```

Abbildung 5.6: MySQL Konfiguration

Ein wichtiger Bean stellt die Konfiguration des ViproWS dar (Abb. 5.7). SpringMVC ist in der Lage, über eine solche Konfiguration zur Initialisierungszeit alle benötigten Vorbereitungen zu treffen, damit ein späterer WS-Aufruf ohne großen Aufwand erfolgen kann:

```
<bean id="vis" class="org.springframework.remoting.jaxws.JaxWsPort..." >
 <property name="serviceInterface" value="arapoport.viproService.Vipro..." />
<property name="wsdlDocumentUrl" val-
ue="http://localhost:8080/viproWS/services/viproService?wsdl" />
    <property name="namespaceUri" value="http://arapoport/viproService" />
    <property name="serviceName" value="viproService"/>
    <property name="portName" value="viproService"/>
</bean>
```

Abbildung 5.7: ViproWS Konfiguration in keida-viproWS-konfiguration.xml

Die Konfigurationsdateien definieren die einzelnen Schichten innerhalb der MVC-Architektur. In den folgenden Unterkapiteln folgt eine ausführliche Beschreibung der Implementation diesen Schichten.

5.3 Controllerschicht (Controller Layer)

Der *DispatcherServlet* ist als Front-Controller aufgebaut. Dieser Front-Controller leitet die Anfragen an den jeweiligen Controller weiter. Damit der *DispatcherServlet* diese finden kann, ist der Package, in dem sie sich befinden, in der xml-Konfigurationsdatei spring-servlet.xml (Abb. 5.2) angegeben. Die Parameter, die die Browseranfragen liefern, sollen dann auch den richtigen Methoden der jeweiligen Controller zugewiesen werden (Abb. 5.9). Dieses "Handler Mapping" wird durch die "annotation-driven"-Definition möglich gemacht (Abb. 5.4). Annotationen wie "*Controller*" oder "*RequestMapping*" dienen dem *DispatcherServlet,* die Controller und die zugehörige Methode auszuwählen (Abb. 5.8).

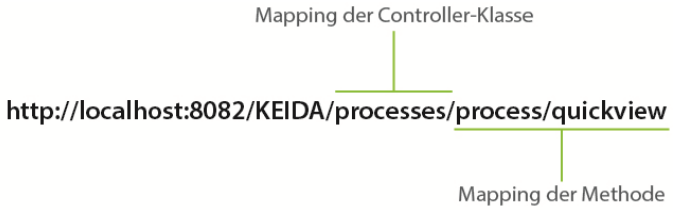

Abbildung 5.8: Handler Mapping

KEIDA ist um die zwei Themen Dashboard und Visualisierungswerkzeug aufgebaut und implementiert Controller, die diese zwei Tatsachen abbilden (Abb. 5.9). Die Browser-Anfragen an das Dashboard werden an den "*Home Page Controller*" sowie "*Process Page Controller*" weitergeleitet. Zuständig für das Visualisierungswerkzeug ist der "*Visualization Page Controller*".

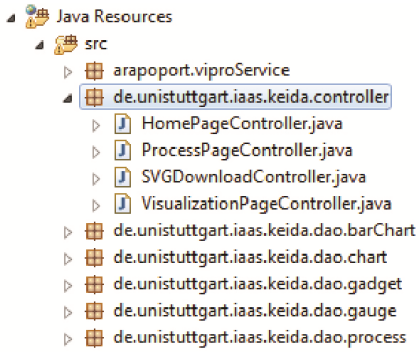

Abbildung 5.9: Controllerklassen

Die Controller sind in der Regel so aufgebaut, dass sie fast keine Funktionalitäten oder Algorithmen implementieren. Sie wissen nur welches Modell für die Bearbeitung der Anfrage zuständig ist und welches View für die Präsentation gebraucht wird. Das wird am Beispiel der Klasse *ProcessPageController* deutlicher (Abb. 5.10). An erster Stelle ist diese Klasse mit *"@Controller*" als Controller annotiert. Sie wird immer dann von dem *DispatcherServlet* ausgewählt, wenn sie in der URL (Abb. 5.8) nach dem Servlet-Namen, in diesem Fall "*processes*", auftaucht. Die Methode, die für eine Visualisierung des Prozesses mit "*id=1*" gebraucht wird, ist mit "*RequestMapping*" mit Wert "*process/quickview*" annotiert. In der Methode wird ein Objekt *ModelAndView* erzeugt, das als Eingabeparameter den Namen des Views bekommt. Das Model liefert in diesem Fall den Namen einer SVG-Datei, die von ViproWS generiert und in einem Ordner auf dem Server abgespeichert wurde. Der *Request-Dispatcher* nimmt diesen Namen, findet das View und schickt eine Antwort zurück an den Browser.

```
@Controller
@RequestMapping(value="/processes")
public class ProcessPageController {
…
        @RequestMapping(value={"process/quickview"}, method=GET)
        public ModelAndView quickView(@RequestParam(value="id", required=false) int
        id)
```

```
        {
          ModelAndView mav = new ModelAndView("processes/process/quickview");
          HashMap hm = this.ppc.getQuickview(id);
          mav.addAllObjects(hm);
          return mav;
        }
….
```

Abbildung 5.10: Mapping einer Controllerklasse

Die Controller liefern in den meisten Fällen den logischen Namen des Views und anhand dieser kann dann die Präsentation bestimmt werden. Die Präsentation ist in JSP-Dateien beschrieben. Das ist aber nicht zwingend. So kann z.B. der Controller "*VisualisationPage*" auch AJAX Anfragen beantworten und liefert ein JSON Objekt (Abb. 5.11). Der Controller "*SVGDownload*" gibt auch keine HTML-Repräsentation zurück, sondern eine SVG-Datei, die sich auf dem Server befindet und vom Benutzer angefragt wird.

```
@RequestMapping(value="/instances",method=GET)
public @ResponseBody List<ProcessInstance> getProces-
sInstancesJSON(@RequestParam("pid") int pid, @RequestParam("date")String date){
      List<ProcessInstance> result = this.vpsc.getProcessInstances(pid, date);
    return result;
}
```

Abbildung 5.11: Mapping einer AJAX Anfrage

5.4 Präsentationsschicht (View Layer)

Beim Aufbau der Präsentationsschicht wurde auf Apache Tiles 2.2[8] gesetzt. Apache Tiles ist ein System für modularen Aufbau von graphischen Oberflächen in Webanwendungen. Die Seiten bestehen in der Regel aus Elementen, die sich immer wieder wiederholen. Apache Tiles dient dazu, diese Elemente, auch Tiles genannt, zu kombinieren und eine komplette Webseite aufzubauen. Die Tiles sind in JSP-Dateien definiert und mittels Beschreibung in xml-Konfigurationsdateien zur Laufzeit auf einer Seite zusammengefasst. Diese Aufteilung der Seiten führt zu einer besseren Organisation der Views. Die KEIDA-Views befinden sich im "*views*"-Ordner unter „*WebContent/WEB-INF*". Die Präsentationsschicht ist in drei Kategorien gegliedert: "home", "processes" und "visualize".

Die Views sind in den *views.xml*-Dateien, die in den jeweiligen Ordnern liegen, definiert. Sie sind hierarchisch aufgebaut. Ganz oben in der Hierarchie steht die in dem Hauptordner liegende *views.xml*-Datei. Sie definiert die Hauptvorlage (*index.jsp*), die den Rahmen jeder HTML-Seite darstellt:

[8] http://tiles.apache.org/

```
<definition name="template" template="/WEB-INF/views/index.jsp">
    <put-attribute name="header" value="/WEB-INF/views/header.jsp" />
</definition>
```

Abbildung 5.12: View Konfiguration in views.xml

Das Attribut “name” (Abb. 5.12) ist der eindeutige Identifikator des Templates. Dieser wird später von den anderen Vorlagen geerbt und erweitert. Das “template“-Attribut enthält den Pfad zur *jsp*-Vorlagedatei. In dieser jsp-Datei werden per benutzerdefinierte Tags “*tiles*” die einzelnen Teile zur Laufzeit eingefügt (Abb. 5.13). Diese Teile sind mit “*put-attribute*” in der *views.xml*-Datei (Abb. 5.12) definiert.

```
<body>
 ...
   <!-- begin header -->
    <tiles:insertAttribute name="header" />
   <!-- end header -->
...
```

Abbildung 5.13: Ausschnitt aus index.jsp

Die Auflösung des Views erfolgt durch den *TilesViewResolver*. Dieser braucht den Namen des Views, der im jeweiligen Controller definiert wird. Der *TilesViewResolver* seinerseits weiß nicht, was eine Tiles-Definition ist und verlässt sich auf dem *TilesConfigurer*. Der *TilesConfigurer* weiß, dass die Views in „*/WEB-INF/views/*“ definiert sind (Abb. 5.14 als Präfix markiert). Die Ant-Style Maske “**” in der Definition (siehe Abb. 5.4) zeigt ihm, dass dort sich ein Ordner mit dem gleichen Namen, der vom *DispatcherServlet* übergeben wird, befindet (in diesem Fall „*home*“, siehe Abb. 5.14) Die *views.xml*-Datei wird ausgelesen und die dort definierte Struktur als Vorlage genommen.

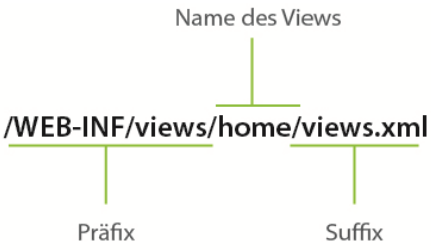

Abbildung 5.14: Auflösung des Namens eines Views

Die Definition des Dashboards sieht so aus:

```
<definition name="home" extends="template">
    <put-attribute name="body" value="myapp.homepage.body"/>
</definition>
<definition name="myapp.homepage.body" emplate="/WEB-INF/views/home/body.jsp">
        <put-attribute name="title" value="/WEB-INF/views/home/title.jsp"/>
        <put-attribute name="datePicker" value="/WEB-
```

```
INF/views/home/datePicker.jsp"/>
        <put-attribute name="kei" value="/WEB-INF/views/home/kei.jsp" />
        <put-attribute name="chart" value="/WEB-INF/views/home/chart.jsp" />
        <put-attribute name="processList" value="/WEB-
INF/views/home/processList.jsp"/>
        <put-attribute name="gadgets" value="/WEB-INF/views/home/gadgets.jsp"/>
</definition>
```

Abbildung 5.15: Definition der Startseite des Dashboards

Beim Aufrufen der URL: "http://localhost:8080/KEIDA/dashboard" im Browser gibt der *HomePageController* als logischen View-Namen "*home*" an den *TilesViewResolver*. Der *TilesViewResolver* wendet sich an den *TilesConfigurer* und er holt die Vorlage mit allen Teilen und gibt sie an den *TilesViewResolver* zurück. Die Information, die vom Model vorbereitet wurde, wird an den richtigen Plätzen eingefügt und die so erstellte HTML-Seite an den Browser zurückgeschickt. Ein Beispielergebnis ist auf Abb. 4.3 zu sehen.

In 5.3 und 5.4 wurde die Funktionsweise der Controller- und View-Schichten beschrieben. Die Geschäftslogik hinter jeder Aktion, die der Controller erlaubt, ist in der Modelschicht implementiert. Das nächste Unterkapitel gibt einen Überblick über die Implementierung dieser Schicht.

5.5 Modellschicht (Model Layer)

Die Modellschicht der serverseitigen Implementierung von KEIDA ist auf den einzelnen Komponenten, die auf jeder Seite des Front-Ends gebraucht werden, aufgebaut. Die Komponenten sind verschiedene Graphiken oder Listen mit Prozessinformationen (siehe Abb. 5.16). Dieser modulare Aufbau führt zu einer klaren Trennung der Funktionalitäten und der Zuständigkeiten bei der Implementierung der einzelnen Komponenten. Der Vorteil dabei ist, dass diese unabhängig voneinander sind. Diese Komponenten lassen sich leicht austauschen, ohne dass andere geändert werden müssen. Das führt auch zu einer Wiederverwendbarkeit der Komponenten.

Die Services, auf denen die einzelnen Seiten aufgebaut sind, tragen den Namen der Controller, die sie benötigen. *ProcessPageController* greift z.B. auf *ProcessPageServiceComposer*. Die einzelnen Services implementieren das Kompositum (Composite) und das Facade -Entwurfsmuster [GHJV11]. Wie das KEIDA Prozessseite Mock-up zeigt (Abb. 5.16), besteht die Seite aus den Komponenten: *Main Chart, Process Information, Key Ecological Indicators* und *Activities*. Jede dieser Komponenten braucht verschiedene Informationen von der Datenbank und stellt eine andere Datensicht dar. Die *ServiceComposer*-Klassen dienen auch als Facade vor dem Controller, weil sie das Benutzen der darunterliegenden Subsystemen (Services) definieren. Diese Subsysteme enthalten die eigentliche Businesslogik.

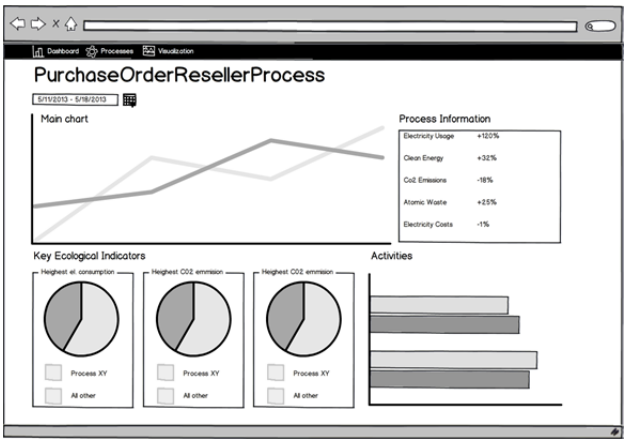

Abbildung 5.16: KEIDA Process page mock-up

Für den Aufbau jeder Graphik werden Informationen, wie die analysierten Indikatoren, die betrachtete Periode, der Name der Graphik usw. gebraucht. Diese Information wird von der Datenbank von einem speziell für diese Aufgabe ausgelegten *ServiceWorker* übernommen. Diese *ServiceWorker* haben einen direkten Zugriff auf die persistente Schicht. Sie wissen, welche Informationen für jeden einzelnen Typ Graphik gebraucht werden. Diese *ServiceWorker* implementieren die Logik, die bei der Bearbeitung dieser Informationen benötigt wird. Nachdem der *ServiceComposer* die Daten bekommen hat, werden sie an das Highcharts-Subsystem übergeben, das für die Generierung der JSON-Objekte zuständig ist. Das Ausgliedern der JSON-Generierung von der Datenbearbeitung führt zu einer klaren Entkoppelung der Zuständigkeiten, um möglichst voneinander unabhängige Subsysteme zu schaffen.

Bei den Text-Komponenten werden nur *ServiceWorker* gebraucht, weil sie keine spezielle Darstellung erfordern. Diese wird direkt vom View übernommen. Im Unterschied dazu stellt das View nur einen Platzhalter für die Graphiken bereit. Die eigentliche visuelle Darstellung ist dem Highcharts JS API überlassen.

Als Beispiel für die oben beschriebene Generierung eines solchen JavaScript Objekts wird der Aufbau der Hauptgrafik erläutert.

Der *ProcessPageController* erhält die Aufgabe, die Übersichtseite des Prozesses mit ID=1 zu zeigen. Nach Eingabe im Browser “ http://localhost:8082/KEIDA/processes/process /view?id=1” wird folgende Methode der Controller-Klasse *ProcessPageController* ausgeführt:

```
@RequestMapping(value={"process/view"}, method=GET)
public ModelAndView processView(@RequestParam(value="id", required=false) int id,
HttpServletRequest request){
    ...
     HashMap hm = this.ppc.getViewPage(sessionRange, sessionPeriod, idI);
     ModelAndView mav = new ModelAndView("processes/process/view");
     mav.addAllObjects(hm);
     return mav;
}
```

Abbildung 5.17: processView()-Methode der ProcessPageController-Klasse

Die Methode übergibt die benötigten Informationen an den *ProcessPageServiceComposer*-Klasse (PPC). Wie am Anfang des Kapitels gezeigt wurde, implementiert die PPC-Klasse das Composite-Pattern. Das bedeutet, dass sie die Struktur der Seite kennt und weiß, welche Komponenten auf der Seite vorkommen und welche Informationen dabei erwartet werden. Die Methode, die in diesem Fall interessant ist, ist *getViewPage* der PPC-Klasse. Sie übernimmt die Aufgabe, eine Komposition mit allen Komponenten zu erzeugen und als ein assoziatives Array (HashMap) zurückzugeben. Für die Vorbereitung der Hauptgraphik werden folgende Schritte benötigt:

```
public HashMap getViewPage(String period, String type, String name){
       ...
      ArrayList<String> indicators = (Ar-
      rayList<String>)this.chartConfiguratorPP.getIndicators().get("mainChart");
      CalendarMath calMath = new CalendarMath(period);
      ArrayList<double[]> data = this.csw.chartDataArray(period, type, indicators,
      name);
      ArrayList<String[]> labels = this.csw.chartLabels(indicators);
      ChartWidget cw = new ChartWidget();
      cw.initialize(calMath.getStartDate(), type, data, labels);
      this.hm.put("highcharts", cw.display());
       ...
}
```

Abbildung 5.18: getViewPage-Methode der PPC-Klasse

Name, Datenreihe, Startdatum für x-Achse und die Legende werden von dem *ChartServiceWorker (csw)* bereitgestellt. Diese werden dem *ChartWidget(cw)* weitergeben. Beim Ausführen der cw display-Methode wird das JS-Objekt geliefert und dieses unter dem Namen “highcharts” in das assoziative Array (*hm*) eingefügt. Im View wird diese ausgelesen und in den Platzhalter, der für die Graphik bereit steht, geschrieben.

Wie schon oben erwähnt implementiert der *ChartServiceWorker* die Geschäftslogik, die hinter jedem Graphiktypen steht. So wird z.B. die Methode *getChartArray()* der *ChartServiceWorker*-Klasse aufgerufen, um die Datenreihe für eine Graphik zu bekommen. Diese Methode liefert die Daten aggregiert nach Periodentyp: pro Stunde, pro Tag usw.

```
private double[] chartMonthlyData(CalendarMath cal, String type, String indicator,
String name){

  List<Month> months = this.chartJDBC.getMonthValue(cal.getStartDate(),
cal.getEndDate(), name, indicator);
    double[] jsData = new double[cal.getMonthsBetween()];
    Calendar temp = Calendar.getInstance();
    if( !months.isEmpty()) {
        for(Month m : months){
```

```
            temp.set(Calendar.YEAR,m.getYear());
            temp.set(Calendar.MONTH, m.getMonth());
            temp.set(Calendar.DAY_OF_MONTH, m.getDay());
            jsData[cal.indexForMonthArray(temp)-1] = m.getValue();
        }
    }
    return jsData;
}
```

Abbildung 5.19: chartMonthlyData()-Methode der Klasse ChartServiceWorker

Die Daten aus der Datenbank werden mit Hilfe vom *JdbcTemplate*-Objekt, das in der *keida-db.xml*- Datei definiert ist, geholt. Das *JdbcTemplate*-Objekt wird zur Initialisierungszeit erzeugt und per Dependencies Injection in der *ChartJDBCTemplate*-Klasse instanziiert. Die *JdbcTemplate*-Klasse übernimmt die Aufgaben, die beim Aufbau der Verbindung mit der Datenbank oder beim Ausführen einer Anfrage an die Datenbank erfüllt werden.

```
public class ChartJDBCTemplate implements ChartDAO {
...
@Autowired
private JdbcTemplate jdbcTemplateObject;
...
public List<Month> getMonthValue(Calendar startDate, Calendar endDate, String name,
String indicatorDefinition){
        String sql = this.buildSQLQuery(startDate, endDate, "month", indicatorDefi-
nition, name);
        List<Month> months = jdbcTemplateObject.query(sql,  new MonthMapper());

        return months;
}
...
}
```

Abbildung 5.20: Ausschnitt aus der Klasse ChartJDBCTemplate

Die Methode *getMonthValue*() der *ChartJDBCTemplate* liefert die Information, die für eine Periode und für einen Indikator in der Datenbank abgespeichert ist. Diese wird dann in *chartMonthlyData*() weiterbearbeitet und als eine Zahlenreihe ausgegeben.

Die gerade gezeigte Struktur steht als Basis für alle *ServiceComposer* und *ServiceWorker*. Die *ServiceComposer* nutzen verschiedene *ServiceWorker* beim Aufbau der Seiten. Die *ServiceWorker* sind dabei so ausgelegt, dass diese unabhängig voneinander sind. Das macht ihren Einsatz in verschiedenen Composer möglich. So wird z.B. die oben gezeigte Hauptgraphik auf der ersten Dashboard-Seite, sowie auf den Seiten mit der einzelnen Prozessübersicht erzeugt.

Die serverseitige Implementierung von KEIDA erfolgt auf Basis der MVC-Architektur. Die dadurch mögliche Trennung der einzelnen Schichten erlaubt eine klare Rollenzuweisung. Die

weitere Aufteilung der Seiten in Komponenten, die wie bei der GUI-Widgets eine eigene Darstellung und Geschäftslogik besitzen, ermöglicht ihr Einsetzen auf mehrere Seiten. Die serverseitige Implementierung ist aber nur ein Teil von KEIDA. Das Frontend in Teil Visualisierungswerkzeug setzt auf einer aktiven Interaktion mit dem Benutzer. Diese Interaktion sowie eine leichte Bedienung des Konfigurationsassistenten erfordern eine Auslagerung von Funktionalitäten in den Browser. Im nächsten Kapitel werden diese näher erläutert.

5.6 GUI Implementierung

Über das Frontend von KEIDA ist der Benutzer in der Lage, alle benötigten Daten, die für die Visualisierung eines Prozesses notwendig sind, einzugeben. Unter benötigten Daten sind die Prozessinstanzen, die Funktionen, die für jeden KEI gebraucht werden, sowie das Mapping (Color, Size usw.) zu verstehen. Dabei gibt es Abhängigkeiten, die zu beachten sind. Diese Abhängigkeiten führen zu bestimmten Konfigurationen und die GUI hat die Aufgabe, den Benutzer dabei aktiv zu unterstützen. Eine wichtige Anforderung jeder Software ist die einfache Bedienung und Erlernbarkeit. Deswegen wurde dieser Teil von KEIDA nicht in separate Schritte aufgeteilt, bei denen eine Anfrage an den Server in jedem Schritt gemacht wird, sondern als eine Desktop-Anwendung konzipiert, die im Browser läuft. Daraus ist der Konfigurationsassistent entstanden. Dieser Wizard wird zum Teil auf dem Server vorbereitet und erfordert auch eine aktive serverseitige Unterstützung.

Die Logik, die den Konfigurationsassistenten steuert, ist in JavaScript implementiert. Erst beim Erfüllen von bestimmten Vorgaben werden Aktivitäten möglich. Diese Ereignisse werden per JavaScript überwacht. So ist z.B. die Bedingung zum Anzeigen des “Visualize”-Buttons die ausführliche Eingabe aller für eine Visualisierung benötigten Informationen. Diese werden beim Klicken auf den Button dem Server mittels eines AJAX-Aufrufs übergeben. Diese AJAX-Aufrufe dienen als Basis des Konfigurationsassistenten und spielen eine große Rolle bei seinem Aufbau. AJAX-Aufrufe ermöglichen es, nur Teile der Webseite von dem Server anzufordern und dadurch ist das Neuladen der ganzen Seite unnötig. AJAX kommt auch beim Laden der Prozessinstanzenliste und bei der zweiten Visualisierung zum Einsatz.

Die Architektur des Front-Endes ist wie auch die Serverseite auf MVC-Basis aufgebaut. Der entscheidende Unterschied dabei ist, dass das View und das Model auf dem Observer-Pattern setzen. Dadurch sind beide immer in Verbindung und das View kann sich entsprechend erneuern, falls das Model neue Informationen zur Verfügung stellt.

Das Dashboard ist auf einer aktiven Unterstützung von JavaScript angewiesen. Die Graphiken, die zu sehen sind, setzen auf der Highcharts-Bibliothek. Die Bibliothek nutzt die Möglichkeiten der modernen Browser und wird auf den neuesten Standards wie HTML5 und SVG aufgebaut. Wie im vorherigen Kapitel gezeigt, muss zuerst die benötigte Information auf dem Server vorbereitet werden, um eine Graphik in die Seite zu integrieren. Die Highcharts-Bibliothek erfordert ein JSON-Objekt, mit dessen Hilfe die Graphik beschrieben wird. Dieses Objekt wird dann von dem Highcharts-API als Vorlage bei der Generierung der svg-Repräsentation, die auf der Webseite zu sehen ist, gebraucht.

6 Anwendungsfall: Optimieren eines Prozesses in KEIDA

In dem DWH, auf dem KEIDA aufgebaut ist, werden Informationen über den Energieverbrauch verschiedener Geschäftsprozesse, die in einem Unternehmen laufen oder deren Ablauf simuliert worden war, gespeichert. In der nachfolgenden Untersuchung werden die sich dort befindenden Informationen, die bei der Simulation zweier Prozesse entstanden sind, als Basis genommen. Ziel dabei ist es, das Optimieren der Ökobilanz eines Prozesses mittels KEIDA zu zeigen.

Beim Optimieren ist es empfehlenswert, die Untersuchung auf dem Dashboard anzufangen. Dort sind alle relevanten Informationen über alle Geschäftsprozesse abgebildet. Auf dem Dashboard sind die Auffälligkeiten sofort zu sehen. So ist es auch auf Abb. 6.1 leicht zu erkennen, dass der „PurchaseOrderResellerProcess“ 98% des Energieverbrauchs ausweist. Die nachfolgende Optimierung wird sich deswegen mit ihm befassen.

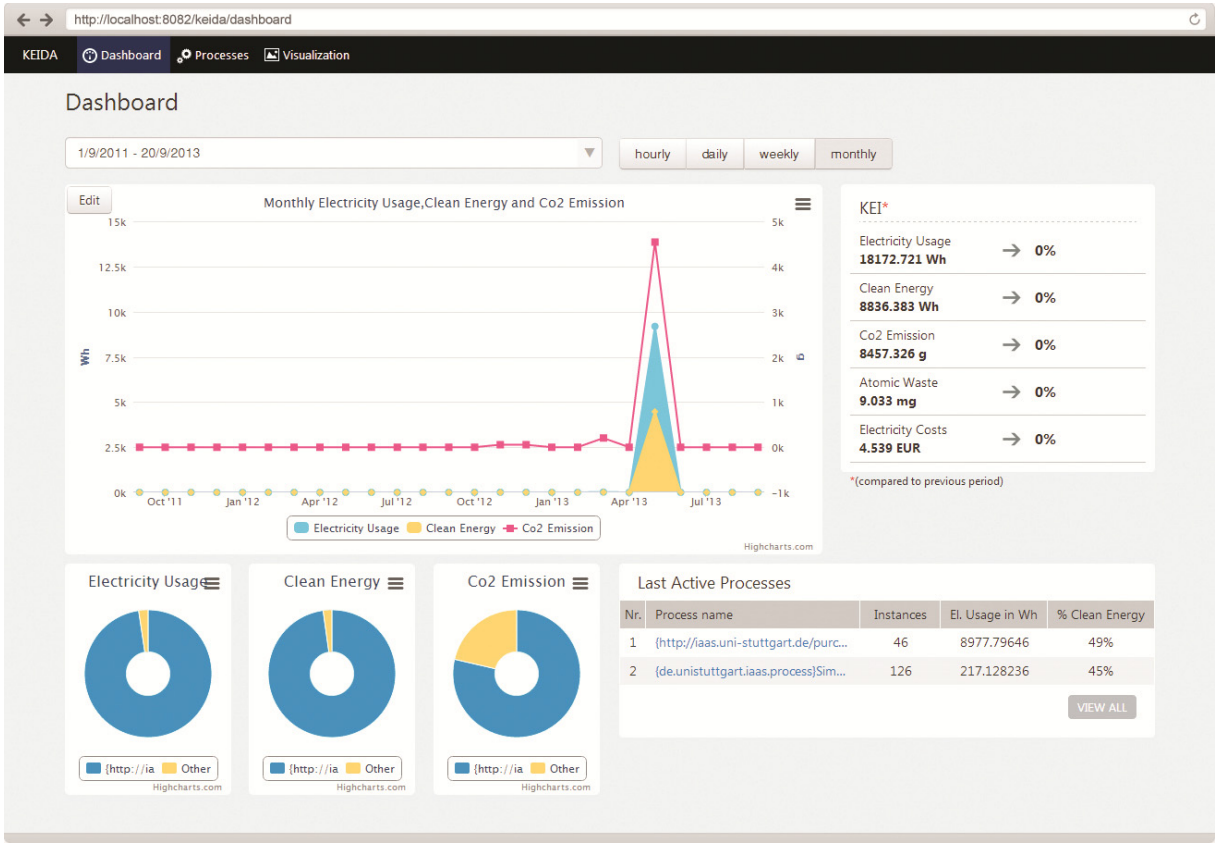

Abbildung 6.1: KEIDA Dashboard

Diese Untersuchung des Prozesses kann auf der einzelnen Prozesssicht-Seite fortgesetzt werden (Abb. 6.2). Hier stehen dem Benutzer verschiedene Informationen zur Verfügung. Es gibt einen gesamten Blick auf alle simulierten Daten über alle Perioden sowie die Möglichkeit, die Periode abzugrenzen und sich nur auf einem bestimmten Zeitraum zu konzentrieren. Die KEI Tachometer bilden den durchschnittlichen Wert jedes KEI bei einem Prozessdurchlauf ab. Auf dieser Seite sind auch die Werte, die die einzelnen Aktivitäten generiert haben, zu sehen. Die Information auf dieser Seite ist immer periodenbezogen. Wählt der Benutzer eine andere

Periode, wird sich diese entsprechend ändern und ist nur dem ausgewählten Zeitraum zugeordnet.

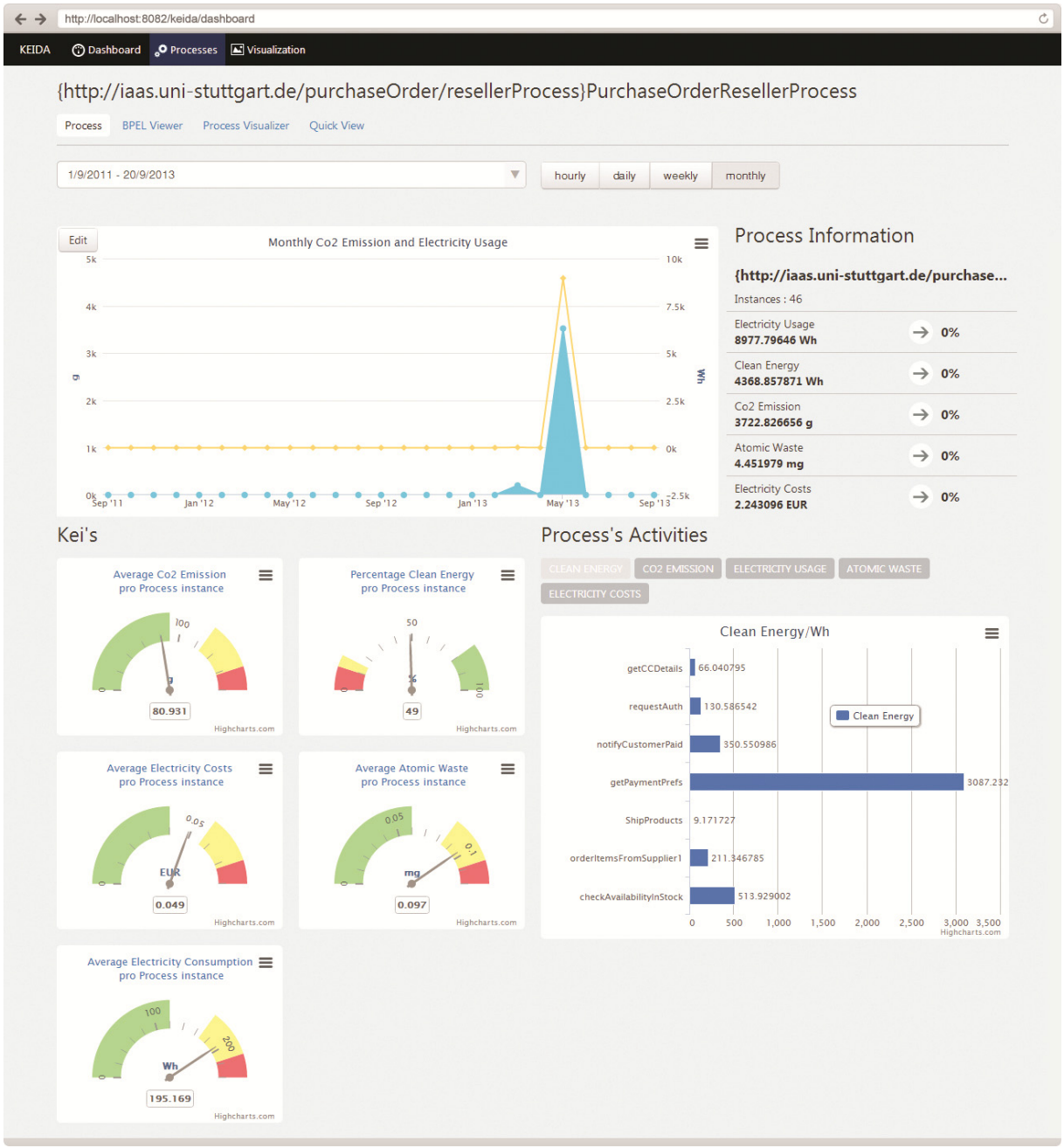

Abbildung 6.2: PurchaseOrderResselerProcess Dashboard

Die KEI-Tachometer zeigen, dass der Energieverbrauch weit über dem gewünschten durchschnittlichen Wert liegt. Rechts davon auf dem Balkendiagramm, bei dem die KEI Werte auf die einzelnen Aktivitäten verteilt sind, kann man sehr schnell fündig werden. Hier ist leicht erkennbar, dass sich der Energieverbrauch mancher Aktivitäten weit über den der anderen erstreckt. Dafür kann es verschiedene Gründe geben, die aus den vorliegenden Informationen nicht nachvollziehbar sind. Das, was man hier erkennen kann, ist, dass die Aktivitäten und insbesondere „checkAvailabilityInStock“ sehr wenig grüne Energie verwenden (Abb. 6.3). Diese Tatsache ist auf dem „Percentage Clean Energy“ sichtbar (Abb. 6.4). Dabei liegt der

Anteil grüner Energie weit unter den gewünschten 80%. Die Priorität der nachfolgenden Optimierung des Prozesses liegt auf der Verringerung des Stromverbrauchs. Damit werden die Senkung des CO_2-Ausstoßes und die Verkleinerung der Mengen Atommüll, die bei der Produktion der Energie anfallen, angestrebt.

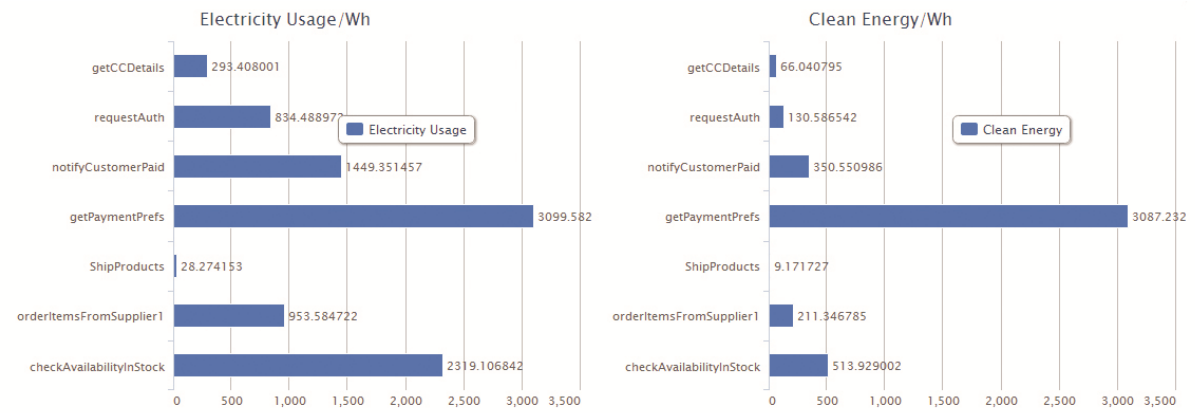

Abbildung 6.3: Verbrauch grüner Energie

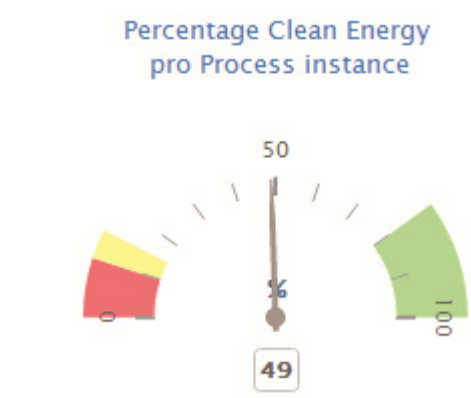

Abbildung 6.4: Durchschnittlicher Anteil der grünen Energie bei einem Prozessdurchlauf

Für die weiteren Untersuchungen wird das Visualisierungswerkzeug eingesetzt, das ein Bestandteil von KEIDA ist und das die graphische Darstellung der Prozessaktivitäten anhand der Informationen, die im DWH liegen und auch mittels externer Informationen möglich macht.

Das Visualisierungswerkzeug besteht aus zwei Teilen. Links befindet sich der Konfigurationsassistent und rechts werden die Visualisierungen angezeigt (Abb. 4.4). Durch ihn sind die Benutzer in der Lage, alle benötigten Informationen einzugeben, um eine Visualisierung zu bekommen. Hier kann man Prozessinstanzen aus einer bestimmten Periode auswählen (Abb. 6.5, links), Funktionen, die einen Zusammenhang darstellen, einfügen (Abb. 6.5, Mitte) und auch die Mapping Methode festlegen (Abb. 6.5, rechts).

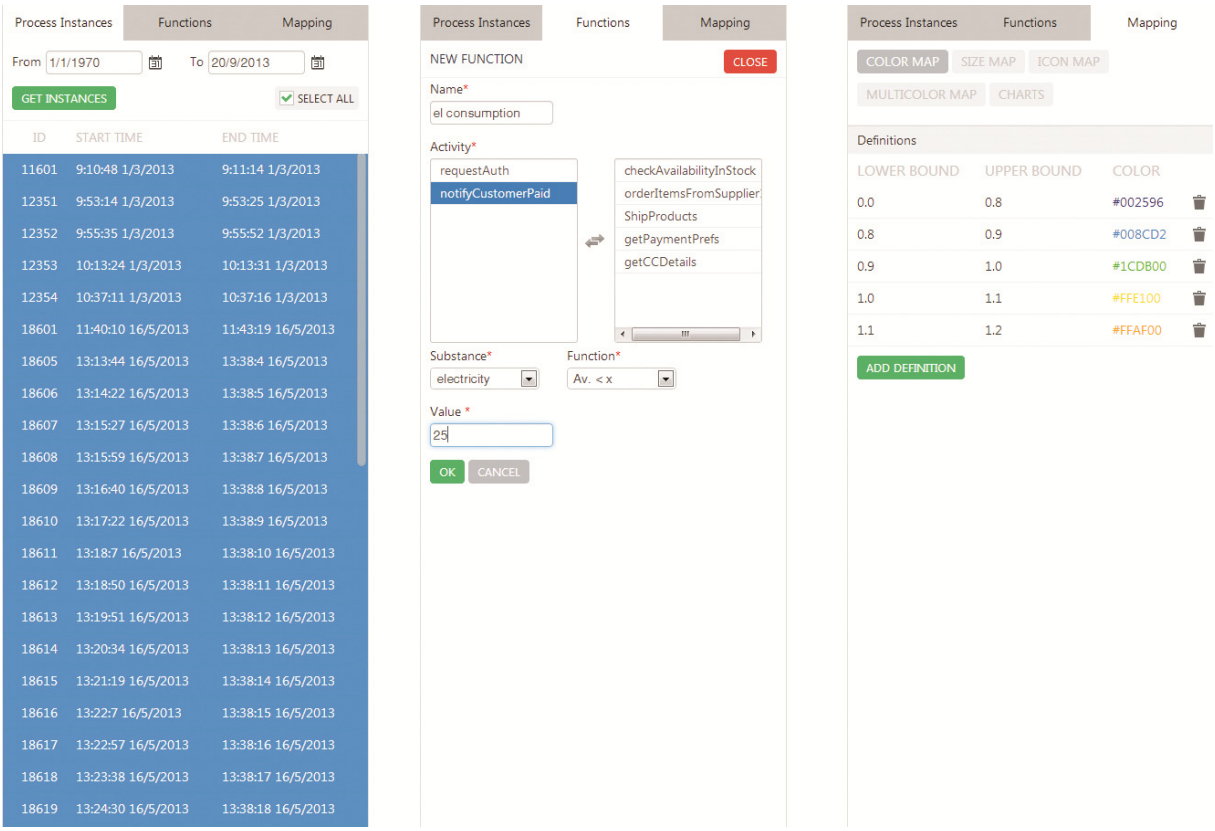

Abbildung 6.5: Konfigurationswizard

Sobald der Benutzer alle Informationen eingegeben hat, kann dieser die Visualisierung vom Server anfordern. Die Konfiguration, die für das gezeigte Beispiel verwendet wurde, ist wie folgt:

- **Prozessinstanzen**. Alle Prozessinstanzen, die im DWH abgelegt sind, wurden ausgewählt. Als Periode wurde der gesamte Zeitraum genommen. Von 1/1/1970 bis heute (20/9/2013).
- **Funktionen**. Um das Beispiel einfach zu halten, wurde nur eine Funktion eigestellt. Dabei wurden alle Aktivitäten genommen, als Substanz wurde Elektrizität ausgewählt und als Funktion durchschnittlicher Verbrauch keiner als der Wert 25 (Wh).
- **Mapping**. Als Mapping wurde „Color Map“ genommen mit den voreingestellten Werten und den zugewiesenen Farben. Z.B. falls der Wert bis zu 80% dem Funktionswert entspricht, wird die Aktivität mit der Farbe #002596 (dunkel blau) gefärbt. Folgende Farbeinstellungen wurden dabei vorgenommen:
    - zwischen 80 und 90% : #008CD2 (hell blau)
    - zwischen 90 und 100%: #1CDB00 (grün)
    - zwischen 100 und 110% : #FFE100 (hell gelb)
    - zwischen 110 und 120%: #FFAF00 (orange)
    - alle Werte, die über 120% des eingegebenen Funktionswerts liegen, werden in dunkel rot gefärbt.

Das Ergebnis kann man auf Abb. 6.6 sehen.

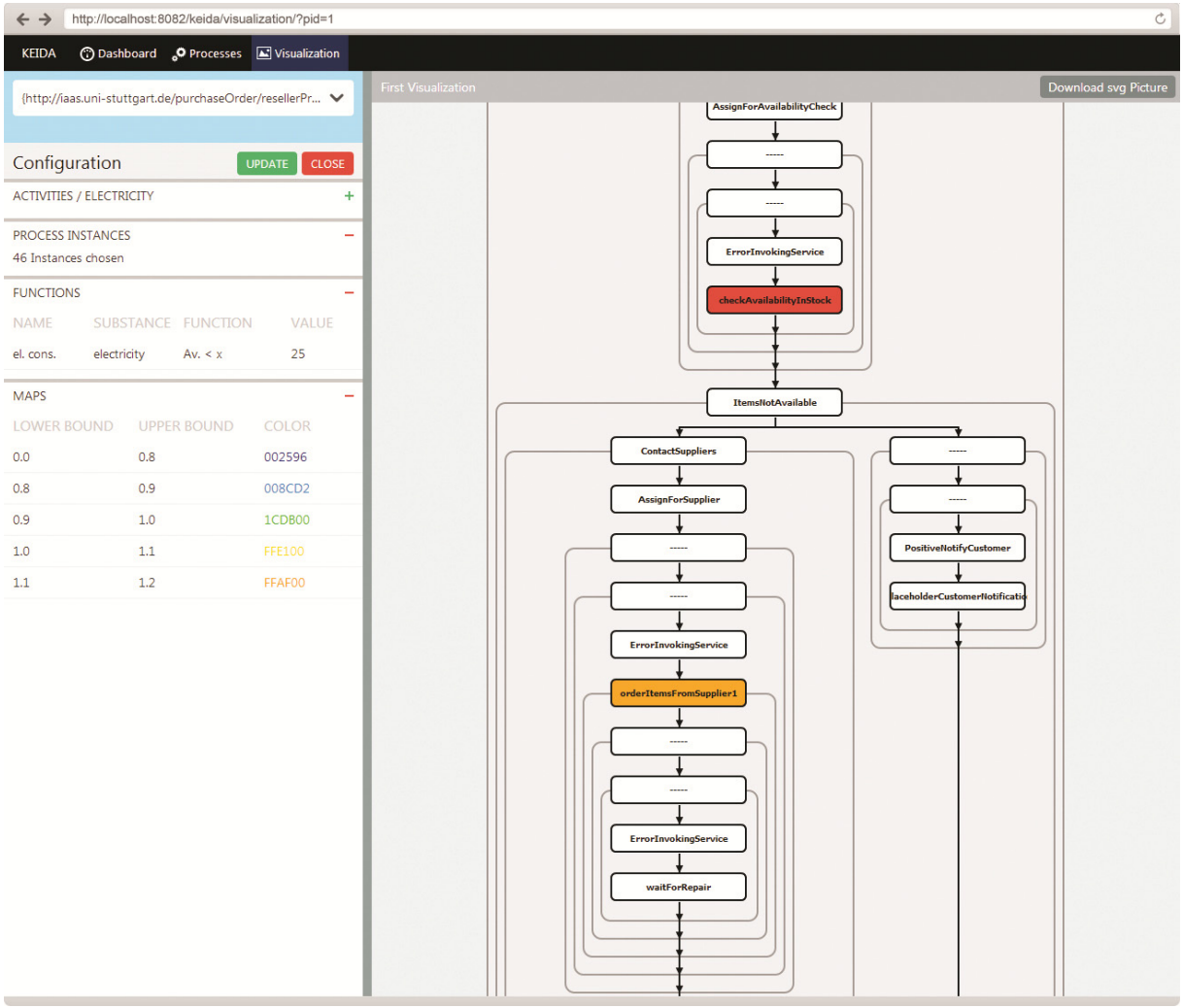

Abbildung 6.6: Visualisierung des PurchaseOrder Prozesses

In diesem ersten Schritt in Richtung einer möglichen Verbesserung der Ökobilanz des Prozesses sind die Aktivitäten, die eventuell später mit energiesparenderen Aktivitäten ersetzt werden sollen, sofort zu sehen. In dem betrachteten Beispiel sind das die „checkAvailabilityInStock“ (Abb. 6.6, in rot) sowie „getPaymentPrefs“ (auf dem Bild nicht zu sehen). Sie haben einen sehr hohen durchschnittlichen Energieverbrauch. „CheckAvailabilityInStock“ weist einen Verbrauch von ungefähr 50Wh pro Durchlauf auf und bei „getPaymentPrefs“ liegt dieser Wert sogar bei 67.38Wh bei einem angestrebten Wert von nur 25Wh. Für die weitere Prozessoptimierung ist es wichtig, dass die Problemaktivitäten sich durch andere, die die gleiche Funktionalität bereitstellen, ersetzen lassen und dass die neuen Aktivitäten bessere Durchschnittswerte haben. Deswegen wird hier angenommen, dass es solche Aktivitäten gibt und sie die jetzigen ersetzen können, ohne die Struktur des Prozesses zu ändern. Für die „checkAvailabilityInStock“ wird angenommen, dass es einen Ersatz gibt mit einem durchschnittlichen Wert von 22Wh und für „getPaymentPrefs“ ist ein Ersatz mit durchschnittlichem Wert von 30Wh zu finden.

Nachdem die neuen Werte eingegeben wurden und die neue Version der Visualisierung angefordert ist, ist sofort zu sehen, dass diese Änderung den gesamten Energieverbrauch um mehr als 32% verringert. Graphisch ist das auf der rechten Seite der Abb. 6.7 auch leicht zu erken-

nen. Die Verfärbung der Aktivität ist (Abb. 6.7, rechts unten) jetzt im hell blauen Bereich (zwischen 80 und 90% des angestrebten Wertes). Im Vergleich mit der Ausgangsituation (Abb. 6.7, oben rechts) ist diese Aktivität dunkel rot gefärbt, was einem viel höheren Wert als den gewünschten entspricht.

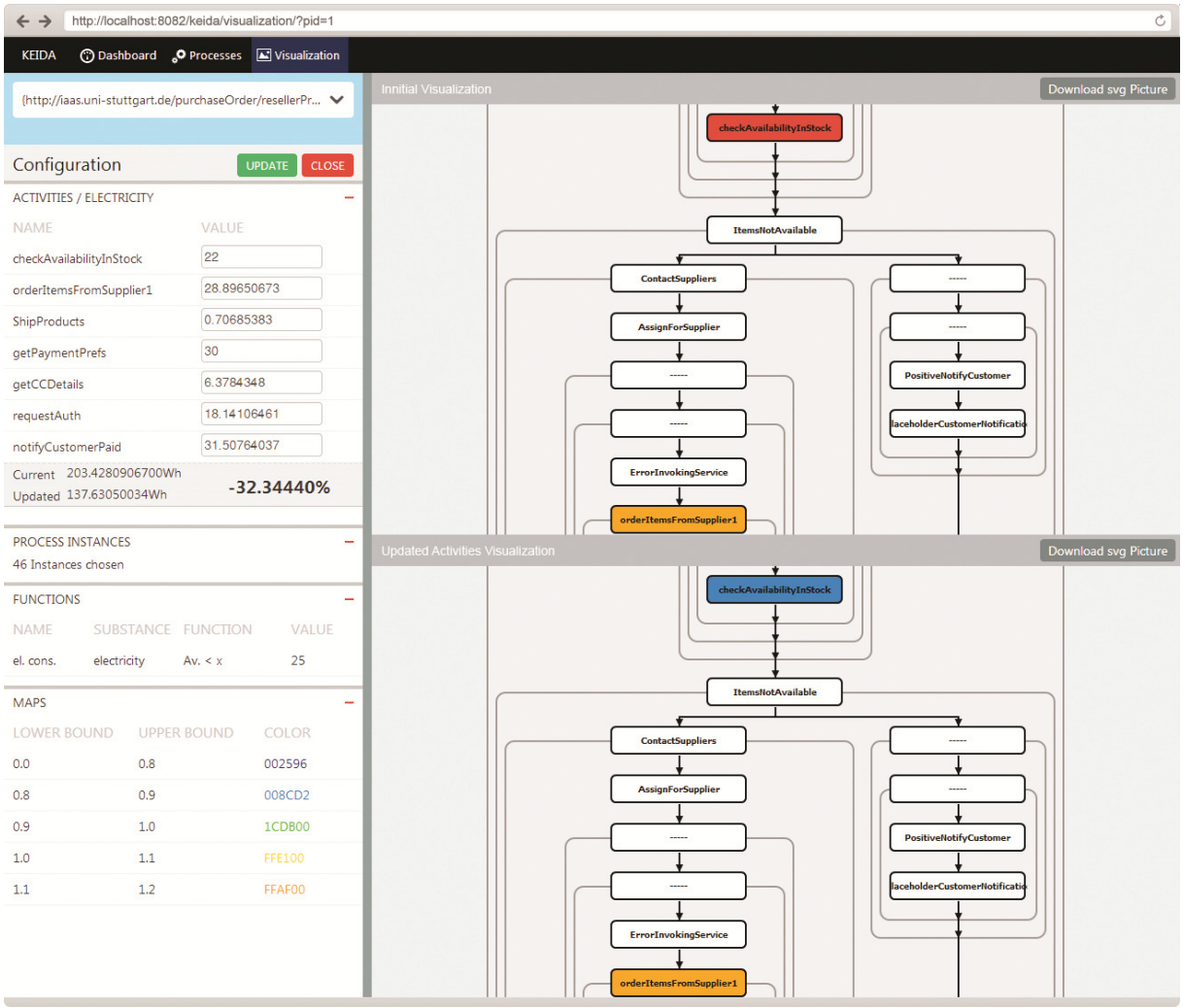

Abbildung 6.7: Vergleich von zwei Visualisierungen

Dieses Beispiel zeigt, wie man mit KEIDA in einigen Schritten bei Vorliegen der benötigten Informationen (z.B. über Aktivitäten, die auch von externen Anbietern stammen können) eine Optimierung eines Geschäftsprozesses bezüglich Energieeffizienz erzielt werden kann. KEIDA stellt die Informationen über das Verhalten des Prozesses graphisch dar und erleichtert dadurch das Ziehen von Rückschlüssen auf die Problemzonen dieses Geschäftsprozesses. Beim Vorliegen externer Daten über die Ökobilanz einer Aktivität lassen sich diese leicht durch die im System abgelegten Daten ersetzen. Dadurch kann schnell visuell gezeigt werden, was für einen Einfluss sie auf dem gesamten Prozess haben. Die so gewonnenen Erkenntnisse kann man später beim Optimieren des Prozesses berücksichtigen, um ein nachhaltiges und mit einer verbesserten Ökobilanz Betreiben dieses Prozesses zu erzielen.

7 Zusammenfassung

Green IT entwickelt sich in den letzten Jahren von einem Trend zu einer neuen Denkweise. Die Unternehmen sehen dabei das große Potential nicht nur Geld zu sparen, sondern ihren Umgang mit den knappen Ressourcen zu ändern. Wie in Kap. 1 gezeigt, machen große Unternehmen wie Google und Facebook die ersten Schritte in diese Richtung und viele kleine Unternehmen wie Strato folgen. Dabei geht es um das Generieren immer neuer Ansätze, wie sich die IT in einem Unternehmen „grüner“ betreiben lässt, sowie darum, durch IT eine Verbesserung des Umgangs mit den knappen Ressourcen zu erzielen.

Software Lösungen wie das KEI Framework stellen nicht nur die Möglichkeiten dar, die Energiefresser unter den Geschäftsprozessen des Unternehmens ausfindig zu machen, sondern geben auch an, welcher Anteil dieser Energie grün ist, wie viel CO_2 in die Atmosphäre ausgestoßen worden ist sowie wie viel Atommüll bei der Nutzung von Kernenergie angefallen ist. Beim Optimieren gibt es verschiedene Lösungen und eine davon stellt KEIDA dar. KEIDA baut auf das KEI Framework auf und versucht, alle relevanten Aspekte des Energieverbrauchs eines Geschäftsprozesses in einer geeigneten Form graphisch dem interessierten Benutzer bereitzustellen. Der Benutzer ist dann in der Lage, wie in Kap. 6 gezeigt wurde, schnell Unregelmäßigkeiten oder Auffälligkeiten aufzuspüren und diese zu untersuchen. Dabei wurde auch gezeigt, wie mit dem Visualisierungswerkzeug schnell und unkompliziert eine graphische Darstellung des Prozesses mittels ViproWS gemacht werden kann. Dabei ist es auch wichtig, dass beim Vorliegen externer Daten über den Energieverbrauch, den CO_2-Ausstoß oder die Mengen Atommüll einer Aktivität, der Benutzer sie mit der bestehenden Aktivität ersetzen und sich graphisch die Unterschiede anschauen kann.

Software Lösungen wie KEI Framework und KEIDA sind eine gute Basis für die Optimierung der Ökobilanz eines Prozesses innerhalb eines Unternehmens. Solche Softwarelösungen werden in der Zukunft in jedem Unternehmen eine große Rolle spielen. Einerseits ist der Sinn und Zweck eines jeden Optimieren eines Prozesses, dass dieser Wettbewerbsvorteile gegenüber der Konkurrenz bringt. Andererseits trägt das Unternehmen dazu bei, die Umwelt zu erhalten und für die künftigen Generationen zu bewahren.

8 Anhang A – KEIDA Konfiguration

1. Webserver

Apache Tomcat Server 7.x.

2. Datenbank Server

MySQL 5.x

3.Konfiguration von KEIDA

Die Konfigurationsdateien befinden sich in dem "*WebContent/WEB-INF*"-Ordner.

Diese sind:

- keida-db.xml
- keida-homepage-configuration.xml
- keida-processPage-configuration.xml
- keida-viproWS-configuration.xml
- keida-services.xml
- keida-visualization-mapping.xml
- spring.servlet.xml

Wie in Kap. 5.2 erläutert wurde, werden in die *spring-servlet.xml* alle anderen Konfigurationsdateien importiert.

3.1. keida-db.xml

Hier kann die Verbindung mit der Datenbank eingestellt werden. Die benötigten Informationen sind: URL der Datenbank, der Username, das Passwort und der Name der Datenbank. Eine Beispielkonfiguration wäre:

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/kei_dwupdate" />
    <property name="username" value="root" />
    <property name="password" value="" />
</bean>
```

Abbildung 8.1: MySQL Konfiguration in keida-db.xml

3.2 keida-homepage-configuration.xml

Diese Datei definiert ein Bean, das die Konfiguration der Graphiken auf dem Dashboard darstellt. Mit dem Bean werden die Indikatoren festgelegt, die auf den Graphiken verglichen werden. Die Indikatoren sind in der Klasse *IndicatorDefinitions* festgelegt.

```
<!--  Indicators:
            - ACTIVITY_EL_USAGE
            - ACTIVITY_CO2_EMISSION
            - ACTIVITY_CLEAN_ENERGY
            - ACTIVITY_ELECTRICITY_COSTS
            - ACTIVITY_ATOMIC_WASTE
          Usage:
            #{indicatorDefinitions.{IndicatorType}}  -->
    <bean id="indicatorDefinitions"
class="de.unistuttgart.iaas.keida.domain.IndicatorDefinitions"/>
    <bean id="chartConfiguratorAll"
class="de.unistuttgart.iaas.keida.service.helper.PageConfigurator">
        <property name="indicators">
            <map>
                <entry key="charts">
                    <list>
 <value>#{indicatorDefinitions.ACTIVITY_CO2_EMISSION}</value>
 <value>#{indicatorDefinitions.ACTIVITY_EL_USAGE}</value>
                    </list>
                </entry>
                <entry key="pies">
                    <list>
 <value>#{indicatorDefinitions.ACTIVITY_CO2_EMISSION}</value>
 <value>#{indicatorDefinitions.ACTIVITY_CLEAN_ENERGY}</value>
 <value>#{indicatorDefinitions.ACTIVITY_ELECTRICITY_COSTS}</value>
                    </list>
                </entry>
            </map>
        </property>
 </bean>
```

Abbildung 8.2: Homepage Konfiguration in der keida-homepage-configuration.xml-Datei

Abb. 7.2 zeigt die schon eingestellte Konfiguration. Diese lässt sich leicht ändern, indem die schon eingegebenen Werte durch neue ersetzt werden oder neue dazugegeben werden. Die Werte, die möglich sind, sind am Anfang der Datei aufgelistet.

3.3 keida-processPage-configuration.xml

Die Graphiken auf den einzelnen Prozesssicht-Seiten lassen sich in der *keida-processPage-configuration.xml* steuern. Die Konfiguration erfolgt wie in 3.2.

3.4 keida-viproWS-configuration.xml

Bei der Visualisierung setzt KEIDA auf das ViproWS. Als Webservice ist ViproWS auf einem externen Server zugänglich. Die Konfiguration des Zugriffs auf das Webservice kann in keida-viproWS-configuration.xml eingestellt werden. Dafür werden folgende Informationen gebraucht (Abb. 7.3): Pfad zur WSDL-Beschreibung, Servicename laut WSDL, Port Name laut WSDL.

```
<bean id="vis"
        class="org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean">
        <property name="serviceInterface"
value="arapoport.viproService.ViproService" />
        <property name="wsdlDocumentUrl"
            value="http://localhost:8080/viproWS/services/viproService?wsdl" />
        <property name="namespaceUri" value="http://arapoport/viproService" />
        <property name="serviceName" value="viproService" />
        <property name="portName" value="viproService" />
    </bean>
```

Abbildung 8.3: ViproWS Konfiguration

3.5 keida-visualization-mapping.xml

In dieser Datei sind die voreingestellten Mappings für das Visualisierungswerkzeug aufgelistet. Sie sind nach Map-Typ gegliedert. Die Konfiguration für das ColorMap ist wie folgt:

```
<entry key="colorMap">
    <list>
        <bean class="de.unistuttgart.iaas.keida.service.helper.MapDefinition">
            <property name="lower" value="0.0" />
            <property name="upper" value="0.8" />
            <property name="color" value="#002596" />
        </bean>
        <bean class="de.unistuttgart.iaas.keida.service.helper.MapDefinition">
            <property name="lower" value="0.8" />
            <property name="upper" value="0.9" />
            <property name="color" value="#008cd2" />
        </bean>
        <bean class="de.unistuttgart.iaas.keida.service.helper.MapDefinition">
            <property name="lower" value="0.9" />
            <property name="upper" value="1.0" />
            <property name="color" value="#1cdb00" />
        </bean>
        <bean class="de.unistuttgart.iaas.keida.service.helper.MapDefinition">
            <property name="lower" value="1.0" />
```

```
            <property name="upper" value="1.1" />
            <property name="color" value="#ffe100" />
        </bean>
        <bean class="de.unistuttgart.iaas.keida.service.helper.MapDefinition">
            <property name="lower" value="1.1" />
            <property name="upper" value="1.2" />
            <property name="color" value="#ffaf00" />
        </bean>
    </list>
</entry>
```

Abbildung 8.4: ColorMap Konfiguration

3.6. keida-services.xml

Hier sind nur die Beans definiert, die in den Klassen innerhalb von KEIDA gebraucht werden. Keine spezielle Konfiguration ist möglich oder erforderlich.

4. Speichern von Visualisierungen

ViproWS liefert die Visualisierung eines Prozesses in Form einer svg-Datei und diese Dateien könnten bis zu 14 MB groß werden. Nachdem diese von dem ViproWS geliefert werden, werden sie in einem speziell für das Ziel ausgelegten Ordner auf dem Server gespeichert. Dann wird der GUI nur der Pfad zu den Dateien weitergegeben und diese werden von dort abgerufen. In dem Prototyp wurde die Abspeicherung so konzipiert, dass jede Visualisierung im „temp"-Ordner des Betriebssystems abgelegt wurde. Nachteil dabei ist, dass dadurch sehr schnell sehr viel Platz belegt wird. Da die Visualisierungen aber immer nur für einen speziellen Fall gemacht werden, die Fälle keine Verbindung miteinander haben und die Zuordnung von alten Visualisierungen nach der Ausgangsituation unmöglich ist, wurde diese Abspeicherung abgeschafft. Beim ersten Visualisierungsvorgang erzeugt KEIDA einen neuen Ordner unter "*Ressourcen*" mit dem Namen "*temp*". Sobald eine neue Visualisierung erzeugt wird, wird die alte gelöscht. Wird eine zweite Visualisierung anhand der ersten gemacht, bleibt die erste erhalten. Damit ist ein nachträglicher Download seitens des Benutzers möglich. Die Bilder, die bereits auf dem Server abgespeichert sind, werden bei einem neuen Visualisierungsvorgang durch neu generierte Bilder ersetzt. Dadurch wird immer nur so viel Platz auf der Festplatte gebraucht, wie die zwei aktuellen Repräsentationen eines Prozesses brauchen.

5. Bibliotheken

Alle externen Java-Bibliotheken, die KEIDA braucht, sind unter "*WEB-INF/lib*" zu finden.

6. Interface Labels

In der Datei "*general.properties*" im Ordner "*WebContent/WEB-INF/resources*" sind alle Beschriftungen, die im Frontend von KEIDA vorkommen, aufgelistet.

9 Anhang B – KEIDA Klassenstruktur

Die Java Klassen, die für die serverseitige Implementierung von KEIDA gebraucht werden, sind wie folgt gegliedert:

- **arapoport.viproService**
- **de.unistuttgart.iaas.keida.controller**
- **de.unistuttgart.iaas.keida.dao.barChart**
- **de.unistuttgart.iaas.keida.dao.chart**
- **de.unistuttgart.iaas.keida.dao.gadget**
- **de.unistuttgart.iaas.keida.dao.gadget**
- **de.unistuttgart.iaas.keida.dao.gauge**
- **de.unistuttgart.iaas.keida.dao.process**
- **de.unistuttgart.iaas.keida.dao.domain**
- **de.unistuttgart.iaas.keida.domain.visualization**
- **de.unistuttgart.iaas.keida.math**
- **de.unistuttgart.iaas.keida.service.composer**
- **de.unistuttgart.iaas.keida.service.helper**
- **de.unistuttgart.iaas.keida.service.worker**
- **de.unistuttgart.iaas.keida.ui.datepicker**
- **de.unistuttgart.iaas.keida.ui.highcharts**
- **de.unistuttgart.iaas.keida.ui.menu**
- **de.unistuttgart.iaas.keida.xml.rtdformat**
- **de.unistuttgart.iaas.keida.xml.rtdinstance**
- **de.unistuttgart.iaas.keida.xml.templatecfg**

1. Webservice-Klassen

arapoport.viproService

Dieses Paket enthält die automatisch generierten Klassen, die beim Aufrufen des ViproWS gebraucht werden.

de.unisttugart.iaas.keida.xml.rtdformat

de.unisttugart.iaas.keida.xml.rtdinstance

de.unisttugart.iaas.keida.xml.templatecfg

Diese Pakete enthalten Klassen, die mit JAXB erzeugt wurden und stellen eine Repräsentation der Konfiguration dar, die an den ViproWS gesendet wird.

2. Controller-Klassen

de.unistuttgart.iaas.keida.controller

- HomePageController

- ProcessPageController

- SVGDownloadController

- VisualizationPageController

Das Paket enthält die Controller-Klassen. Diese Controller sind mit "*Controller*" annotiert und enthalten alle Methoden, die zusätzlich mit "*RequestMapping*" annotiert und auf spezifische Funktionen zugeschnitten sind. Die Methoden machen nur einen Aufruf des zugehörigen Modells und legen die richtige Präsentation fest. (Abb. 9.1)

```
@RequestMapping(value={"/","/dashboard"}, method=RequestMethod.GET)
    public ModelAndView showDashboard( HttpServletRequest request) {

        LinkedHashMap hm = this.hmsi.getModel("","");

        request.getSession().setAttribute("dateRange", "");
        request.getSession().setAttribute("radio", "");

        ModelAndView mav = new ModelAndView("home");

        mav.addAllObjects(hm);
        return mav;
    }
```

Abbildung 9.1: showDashboard() - Methode der Klasse HomepageController

Der **HomePageController** implementiert die Methoden *showDashboard* und *updateFirstPage*. Beide haben fast die gleiche Funktion, der Unterschied liegt darin, dass die erste auf eine GET Anfrage zugeschnitten ist und die zweite Antworten an POST Anfragen seitens des Browsers liefert. Das *showDashboard* ermöglicht eine Repräsentation der Daten, ohne dabei eine spezielle Periode in der GUI anzugeben, wobei die zweite das Start- und Enddatum der Periode bekommt, die der Benutzer ausgewählt hat.

ProcessPageController implementiert die Methoden, die einer Suche nach einem bestimmten Prozess, einer Visualisierung, einer Anfrage der BPEL-Datei sowie der Darstellung des Dashboards des ausgewählten Prozesses, antworten.

SVGDownloadController implementiert nur eine einzige Methode, die eine SVG-Datei liefert. Der Controller ist dafür da, weil die SVG-Dateien im Browser als Bilder angezeigt werden. Damit aber diese als Download bereitgestellt werden können, muss die Antwort an

den Browser modifiziert werden und der Header als "*attachment; filename=SVGDATEINAME.svg*" gekennzeichnet werden. Dadurch wird die Datei im Browser als Download bereitgestellt und nicht direkt angezeigt.

VisualizationPageController implementiert Methoden, die statt HTML JSON Objekte liefern. Sie erhalten von der Anfrage JSON Objekte, die vom JavaScript auf dem Klient generiert wurden. Als Antwort wird wieder ein JSON Objekt zurückgeschickt.

3.Modell-Klassen

de.unistuttgart.iaas.keida.service.composer

de.unistuttgart.iaas.keida.service.helper

de.unistuttgart.iaas.keida.service.worker

Die drei Pakete enthalten die Klassen, die das Modell in der MVC-Architektur bilden. Die Composer Klassen implementieren die Struktur der einzelnen Seiten. Sie sind auf Basis des Kompositum Entwurfsmusters aufgebaut. Die Methoden dieser Klassen liefern ein vollständiges Model der zugehörigen Seite oder Unterseite.

de.unistuttgart.iaas.keida.service.composer

- **HomePageServiceComposerImpl**
- **ProcessesPageServiceComposerImpl**
- **VisualizationPageSerivceComposer**
- **PageServiceComposer**

Die Namen der Klassen entsprechen dem Namen der Controller-Klasse, von der sie aufgerufen werden. **HomePageServiceComposerImpl** enthält z.B. die Methoden, die das Modell bei der Generierung der Daten der Dashboard-Seite braucht. Die anderen Klassen haben eine ähnliche Funktionalität. Sie entspricht den Anforderungen der anderen Seite. **PageServiceComposer** ist eine abstrakte Klasse, die von den *„*.Impl***"**-Klassen geerbt und erweitert wird.

de.unistuttgart.iaas.keida.service.helper

- **SerivceCaller**
- **ServiceConfiguration**
- **SvgFileWriter**
- **TemplateHelper**

Das sind alle Klassen, die vom *ServiceComposer* oder *ServiceWorker* gebraucht werden. *ServiceCaller* ist für das Aufrufen des ViproWS zuständig, *ServiceConfiguration* für die Konfiguration dieses Aufrufs. *SvgFileWriter* speichert die zurückgegebene svg-Repräsentation auf dem Server. *TemplateHelper* arbeitet zusammen mit der *ServiceConfiguration-Klasse*.

de.unistuttgart.iaas.keida.service.worker

- BarChartServiceWorker

- ChartServiceWorker

- GadgetServiceWorker

- GaugeServiceWorker

- KEIIndicatorServiceWorker

- ProcessServiceWorker

- ServiceWorker

-VisualizationServiceWorker

Diese Klassen implementieren die Geschäftslogik innerhalb des Modells. Sie instanziieren ein *JdbcTemplate*-Objekt und sind dadurch in der Lage, Daten aus dem DWH zu holen. Die Klassen tragen die Namen der Komponenten, aus denen eine Seite im Frontend aufgebaut ist. *BarChartServiceWoker* implementiert z.B. die Logik, die hinter den Daten einer Balkengraphik steckt.

4.DAO-Klassen

de.unistuttgart.iaas.keida.dao.chart

de.unistuttgart.iaas.keida.dao.gadget

de.unistuttgart.iaas.keida.dao.gadget

de.unistuttgart.iaas.keida.dao.gauge

de.unistuttgart.iaas.keida.dao.process

de.unistuttgart.iaas.keida.dao.domain

Die Gruppe der Data Access Objekte (DAO) stellt die Ansammlung von Klassen, die die Ergebnisse der Anfragen an das DWH repräsentieren. In jedem Paket ist eine **.JDBCTemplate* Klasse zu finden. Diese Klasse enthält die eigentlichen Queries und liefert je nach Anfrage eine Liste von Objekten.

```
public List<Day> getDayValue(Calendar startDate, Calendar endDate, String name,
String indicatorDefinition){

    String sql = this.buildSQLQuery(startDate, endDate, "day", indicatorDefinition,
name);
    List<Day> days = jdbcTemplateObject.query(sql, new DayMapper());

    return days;
}
```

Abbildung 9.2: getDayValue()-Methode der Klasse chartJDBCTemplate

Die getDayValue()-Methode liefert z.B. eine Liste mit den Daten aller Prozesse aggregiert pro Tag. Diese Information wird dann benötigt, wenn als Periodentyp „Tag“ ausgewählt wurde.

Zu jeder DAO Klasse existiert auch eine *Mapper*-Klasse. Sie wird von der *JDBCTemplate*-Klasse gebraucht, damit die Ergebnisse der DWH-Anfragen richtig gemappt werden können.

5.Math-Klassen

de.unistuttgart.iaas.keida.math

- **CalendarMath**
- **MappingMath**

Das Paket enthält die Klassen *CalendarMath* und *MappingMath*. Die Klasse *CalendarMath* implementiert Methoden, die mit Daten (z.B. 09/01/1970-09/01/2013) rechnen. Die Klasse berechnet z.B. wie viele Tage, Wochen und Monate es innerhalb einer Periode gibt. Die *MappingMath*-Klasse dagegen wird bei der Konfiguration der Daten gebraucht, die an das ViproWS gesendet werden. Jede Mapping-Art erfordert eigene Berechnungen und die Methoden dieser Klasse stellen diese Funktionalitäten zur Verfügung.

6.UI-Klassen

de.unistuttgart.iaas.keida.ui.datepicker

de.unistuttgart.iaas.keida.ui.highcharts

de.unistuttgart.iaas.keida.ui.menu

Die Klassen, die in diesen Paketen definiert sind, helfen beim Aufbau von verschiedenen auf den Frontend-Seiten auftauchenden UI-Elementen. Das Menü-Paket dient z.B. dem Aufbau des Menüs.

Das **highcharts**-Paket enthält die Klassen, die die JSON Repräsentation einer Graphik im Format der **Highcharts**-Bibiliothek liefern.

Das **datepicker**-Paket dagegen liefert die JavaScript, die für den Aufbau der Datepicker-Widget zuständig ist.

7.Domain-Klassen

de.unisttugart.iaas.keida.domain.visualization

In diesem Paket sind die Klassen definiert, die bei der AJAX aufgerufen im VisualizationPageController gebraucht werden. Die JSON-Objekte, die an KEIDA geschickt werden, müssen beim Eintreffen in der zuständigen Methode eine entsprechende Java-Repräsentation haben.

Literaturverzeichnis

[Al03] Alur, Deepak: Core J2EE Patterns, Prentice hall Professional, 2003

[Bmu11] Bundesministerium für Umwelt, Naturschutz und Reaktorsicherheit: Papierverbrauch in Deutschland, http://goo.gl/K3KkZl, besucht am 05.09.2013

[BES10] Buchta, Dirk; Eul, Marcus; Schulte-Croonenberg, Helmut: Strategic IT-Management: Increase value, control performance, reduce costs, Springer, 2010

[BS12] vom Brocke, Jan; Seidel, Stefan: Green Business Process Management: Towards the Sustainable Enterprise, Springer, 2012

[CW10] ComputerWeekly: Carbon reduction brings financial benefits, http://goo.gl/yKTFCg, besucht am 04.09.2013

[DS90] Davenport, Thomas; Short, James: The new industrial engineering: Information technology and business process redesign, in Sloan Management Review, Vl. 31, No.4, Massachusetts Institute of Technology, 1990

[Eg13] Eggert, Ulrich, Wachstum im Handel durch Internet, E-Commerce & Co., http://goo.gl/9DRwU3, Besucht am 02.09.2013

[Fe04] Few, Stephen: Dashboard Confusion, Perceptual edge, http://goo.gl/o6cYLU, 2004

[GHJV11] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John: Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software, Addison-Wesley, 2011

[Go13] Google, http://www.google.com/green/bigpicture/, besucht am 03.09.2013

[IT13] ITWissen: J2EE Komponentenmodell, http://goo.gl/ZNytpg, besucht am 10.09.2013

[In96] Inmon, William: Building the Data Warehouse, John Wiley, 1996

[Jo12] Jones, Terell: What Are the Top 10 Green IT Strategies?, http://goo.gl/A78kf2, besucht am 04.09.2013

[Joh12] Johnson, Scott: Introduction to Data Deduplication in Windows Server 2012, http://goo.gl/Hdtebf, besucht am 05.09.2013

[KMU06] Kemper, Hans-Georg; Mehanna, Walid; Unger Carsten: Business Intelligence: Grundlagen und Praktische Anwendungen, Westdeutscher Verlag, 2006

[Kr13] kra, Suchmaschinenriese: Google verbraucht so viel Strom wie eine Großstadt, http://goo.gl/ueYxIs, Besucht am 02.09.2013

[KW10] Kosch, Bernd; Wagner, Heinz: Alles im grünen Bereich – Mit Green IT zu Energieeffizienz und Nachhaltigkeit, in Green Office, Gabler, 2010

[LN11] Loos, Peter; Nebel, Wolfgang: Green IT: Ein Thema für die Wirtschaftsinformatik?, in WIRTSCHAFTSINFORMATIK, Vol.53, Nr.4, SP Gabler Verlag, 2011

[MB08] Martinez, Nathaneil; Bahloul, Karim: European Organisations and the Business Imperatives of Deploying a Green and Sustainable IT Strategy, http://goo.gl/DU8Ebc, IDC, 2008

[Ms08] Murugesan, San: Harnessing green IT: Principles and Practices, in IT Pro Jan/Feb 2008, http://goo.gl/yZoi8v, IEEE Xplore, 2008

[NB13] Netbeans, The NetBeans E-commerce Tutorial - Designing the Application, http://goo.gl/FH2Kau, besucht am 02.09.2013

[NLM11] Nowak, Alexander; Leymann, Frank; Mietzner, Ralph: Towards Green Business Process Reengineering, Universität Stuttgart, Deutschland, 2011

[Ra13] Rüdiger, Ariane: Die richtige Kühlung für das Rechenzentrum, http://goo.gl/9OJ2l3, besucht am 05.09.2013

[RC04] Rivard, Kurt; Cogswell, Doug: Using Analytical Dashboards to cut through the clutter, in DM Review, http://goo.gl/hle9Wc, April 2004

[Rm11] Rouse, Margaret: business process management (BPM), http://goo.gl/YRVbPL , besucht am 03.09.2013

[Sk11] Summitt, Krista: The four Pillars of BPM 7.5 Part Two: Governance, http://goo.gl/EB1s6r, besucht am 03.09.2013

[SSJ02] Singh, Inderjeet; Stearns, Beth; Johnson, Mark: designing Enterprise Applications with the J2EE Platform, Second Edition, Pearson, 2002

[St08] Stern: It-Branche gibt sich umweltfreundlich, http://goo.gl/bp2XtU, besucht am 03.09.2013

[UN12] United Nations ESCAP, Low Carbon Green growth Roadmap for Asian an the Pacific: Fact Sheet – Green technology, http://goo.gl/2NTnsF

[Va12] Ventarman, Archana: Global census shows datacentre power demand grew 63% in 2012, http://goo.gl/bwFGsA, besucht am 03.09.2013

[Ve10] Verdanti: Carbon Disclosure Project Study 2010: The telepresence Revolution, http://goo.gl/TgJmQv

[Wa11] Walls, Craig: Spring in Action, Third Edition, Manning Publications, 2011

[WMC] Workflow Management Coalition, The Workflow Management Coalition Specification, http://goo.gl/dwULJU, besucht am 02.09.2013

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben. Wörtliche und sinngemäße Übernahmen aus anderen Quellen habe ich nach bestem Wissen und Gewissen als solche kenntlich gemacht.

Stuttgart, den 11. Oktober 2013 _____