Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Erstellung eines Web-Editors für Informationsverarbeitungsketten unter Zuhilfenahme von GWT

Steven Reim
Matrikelnummer: 4278940
steven.reim@fu-berlin.de

Betreuer: Dr. Edzard Höfig
Eingereicht bei: Prof. Dr. Ina Schieferdecker
Zweitgutachten: Prof. Dr. Elfriede Fehr

Berlin, 30. April 2014

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

30.04.2014

Steven Reim

Steven Reim

Zusammenfassung

Informationen sind allgegenwärtig und weitreichend. Sie entstehen, werden gesichtet, manipuliert und verbreitet. Möchte man anschaulich abbilden, woher eine Information kommt, welche Wandlungen sie vollzogen hat, an welchen Stellen sie wiederverwendet und verändert worden ist, bis hin zum aktuellen Gebrauch der Information, entstehen zumeist sehr lange und komplexe Ketten.
Hier ist auch von sogenannten Informationsverarbeitungsketten die Rede.
Im Rahmen dieser Bachelorarbeit wird genau diese Problematik untersucht und versucht, eine übersichtliche, visuelle Möglichkeit zu schaffen, um den Fluss von Informationen abzubilden und auch aufzubauen.
Hierzu wird eine Web-Plattform entwickelt, welche in der Lage ist, generisch verschiedenste Komponenten einer Informationsverarbeitungskette zu erstellen und zu verarbeiten.
Dies wird realisiert mittels des *Google Web Toolkits* (GWT) und der für Drag&Drop-Funktionalität unterstützenden Bibliothek *kineticGWT*.

Inhaltsverzeichnis

1 Einführung **1**
1.1 Einleitung 1
1.2 Problemstellung und Motivation 1
1.3 Aufbau der Arbeit 2

2 Grundlagen **4**
2.1 Informationsverarbeitungsketten 4
2.1.1 Information 4
2.1.2 Ontologie 5
2.1.3 Web 6
2.1.4 Semantisches Web 8
2.1.5 Resource Description Framework 9
2.1.6 Web Ontology Language 11
2.1.7 Linked Open Data 12
2.1.8 Informationsverarbeitungsketten 12
2.2 Google Web Toolkit 13
2.2.1 Eigenschaften 13
2.2.2 Statistiken 18

3 Aggregation von Informationen - Eine Recherche **22**

4 Konzeptionierung **26**
4.1 Anforderungen 26
4.2 Vorbedingungen 28
4.3 Architektur 29

5 Umsetzung **33**
5.1 Verwendete Technologien 33
5.1.1 Grundsätzliche Werkzeuge 33
5.1.2 Weitere Frameworks und Bibliotheken 33
5.2 Implementierung 34
5.2.1 Editor 35
5.2.2 Komponenten 37
5.2.3 Verwaltungsoberfläche 41
5.2.4 User-Management und Workspaces 43
5.2.5 Internationalization 45

6 Evaluierung **47**
6.1 Validierung der Anforderungen 47
6.2 Generizität der Komponenten 48
6.3 Nutzbarkeit 49

7 Fazit **51**
7.1 Überblick 51
7.2 Ausblick 51

Abbildungsverzeichnis

1 Austausch von Informationen 4
2 Beispielhafte Ontologie [17] 6
3 Weltweit erste Webseite, welche im Rahmen des CERN-Projekts entstanden ist 7
4 Semantic Web Stack[38] 9
5 RDF-Statement 10
6 RDF-Statement in XML 10
7 Panel in GWT - Java 15
8 Panel in GWT - UiBinder 15
9 Beispiel einer JSNI-Methode 16
10 Beispiel eines GWT-Button-Events 16
11 Ein Event in GWT 17
12 Feuern eines Events in GWT 18
13 Registrieren eines Events in GWT 18
14 Google Web Toolkit - Usage Statistics [7] 19
15 Google Web Toolkit - Verteilung auf Themenbereiche [7] . . . 20
16 Google Web Toolkit - Benefits [13] 21
17 Beispielhafte Pipe in Yahoo Pipes 23
18 Beispielhaftes Ergebnis einer Pipe 24
19 Mangold-Architektur im UML-Diagramm 32
20 Mangold - Editor Übersicht 35
21 Mangold - Navigation Components 36
22 Beispiel-Komponente in JSON 38
23 Die InputOutputComponent 40
24 Die Verwaltungsoberfläche 41
25 Beispielhafte Baumdarstellung einer MySQL-Komponente . . 42
26 Beispielhafter Prozesscode einer Pseudo-MySQL-Komponente 43
27 Das I18N-Interafce 46

1 Einführung

1.1 Einleitung

Der Mensch ist dauerhaft umgeben von Informationen. Er stößt auf sie, indem er E-Mails liest, er nimmt sie wahr beim Besuch auf seinem bevorzugten Sozialen Netzwerk, er ist auf der Suche nach ihnen beim Surfen durch verschiedene Weblogs und er kommt selbst beim Online-Shoppen nicht drumherum neue Informationen aufzunehmen.

Bei dieser wahren Flut an Informationen, die einen Menschen überkommt, fällt es generell nicht mehr leicht, sie zu filtern und zu analysieren. Wollen wir den Ursprung einer Informationsquelle herausfinden, etwa um weiteres diese Information betreffendes Wissen zu erhalten oder um Rückschlüsse auf den Wahrheitsgehalt der Information ziehen zu können, sind wir nicht selten auf umfangreiche und langatmige Recherchearbeit angewiesen. Dies gestaltet sich besonders schwierig aufgrund des Sachverhalts, dass Informationen zumeist bei ihrem Aufgreifen manipuliert werden, bevor sie weitervermittelt werden.
Nehmen wir als Beispiel einer Information eine aktuelle Nachricht einer großen Online-News-Plattform. Diese Nachricht wird von diversen anderen News-Seiten, Zeitungsverlagen oder Blog-Betreibern entdeckt und auf ihrer eigenen Plattform publiziert - umformuliert, gekürzt oder mit anderen Quellen kombiniert.
Dieser Prozess der Weitervergabe einer Information über das Aufnehmen und Verarbeiten dieser bis hin zum Leser wird als *Informationsverarbeitungskette* bezeichnet.

Diese Arbeit hat das Ziel sich genau diesem Thema anzunehmen. Im Rahmen dieses Projektes soll ein Prototyp für eine Anwendung entworfen werden, welcher umfangreiche Informationsketten nachstellen und eigene Informationsflüsse produzieren kann. Es soll einem Endanwender möglich gemacht werden, mittels einer einfachen Bedienung dieser Anwendung selbst Informationen beschaffen und weiterverarbeiten zu können, ohne mühsam Systeme dafür entwickeln zu müssen.

1.2 Problemstellung und Motivation

Woher kommen Informationen? Wo werden sie eigentlich von wem wiederverwendet? Und wie können wir Informationen für unsere Zwecke nutzen?
Die Herkunft von Informationen zu ergründen und deren Werdegang nachzuvollziehen gestaltet sich in den meisten Fällen als nahezu unmöglich. Freilich existieren spätestens seit der Prägung und Entwicklung der Forschungsrichtung des Semantischen Web (Kapitel 2.1.4) zahlreiche Ansätze, um den Fluss von Informationen nachvollziehbarer werden zu lassen. So ist etwa das *Re-*

source Description Framework (Kapitel 2.1.5) ein sehr umfangreiches und nützliches Hilfsmittel, um Rückschlüsse auf Herkunft und Art der Verwendung von Informationen ziehen lassen zu können.
Jedoch werden solche Ansätze nur von den wenigsten Webseiten-Betreibern genutzt. Zudem benötigt diese Art der Präsentation von Informationsflüssen stets noch umfangreiche Arbeit zur visuellen Aufbereitung, um die Wege von Informationen übersichtlich und lesbar darstellen zu können.

Möchte man noch einen Schritt weiter gehen und sich selbst automatisiert Informationen besorgen und gefiltert aufbereiten um eigene neue Ontologien zu bilden, stellt sich dies ebenso schwierig dar. Hilfsmittel zur gezielten Suche nach Informationen und deren anschließenden Verarbeitung und Darstellung sind auf dem Markt sehr rar. Es ist nicht ohne weiteres möglich, sich eine Kette von Verarbeitungsprozessen anhand verschiedener Quellen aufzubauen, welche Ergebnisse übersichtlich an einem gewünschten Ort platziert und zudem noch wart- und bearbeitbar ist.

1.3 Aufbau der Arbeit

Im folgenden Abschnitt werden auf die fachlich und technisch relevanten Grundlagen erläutert und eingangs geklärt, was wir eigentlich genau meinem, wenn wir im Kontext der Informatik von Informationen (Kapitel 2.1.1) und Ontologien (Kapitel 2.1.2) sprechen. Darauf aufbauend werden die zum vollen Verständnis der Informationsverarbeitungsketten (Sektion 2.1) notwendigen Begriffe wie *Semantisches Web* (Kapitel 2.1.4), *RDF* (Kapitel 2.1.5) und *Linked Open Data* (Kapitel 2.1.7) geklärt, bevor wir uns der Erläuterung der Informationsverarbeitungsketten (Kapitel 2.1.8) selbst zuwenden.
Folgend wird Wert auf die Klärung der genutzten Technologie gelegt und umfangreich das Thema Google Web Toolkit (GWT, Kapitel 2.2) behandelt.

Das dritte Kapitel (Aggregation von Informationen - Eine Recherche) beinhaltet die Analyse einer bereits bestehenden Technologie, welche dazu verwendet wird, Informationen verschiedener Quellen zu aggregieren und zu modifizieren und an separater Stelle gesondert auszugeben, bzw. zu modifizieren.
Kapitel 5 beschreibt die eigentliche Umsetzung des Anwendung, im Folgenden *Mangold* genannt. Nach der Erarbeitung eines Projektdesigns, in welchen speziell Fokus auf die Generizität von Komponenten Wert gelegt wird (Kapitel 4.1), wird im Teil der Umsetzung Bezug auf die einzelnen großen Bestandteile des Editors genommen - Im Speziellen auf den Editor, den Komponenten, dem User-Management, sowie der Verwaltungsoberfläche (Kapitel 5.2.1, 5.2.2, 5.2.4 und 5.2.3).

Im Anschluss wird im Kapitel 6 das Ergebnis der Arbeit analysiert und geprüft, inwieweit das Ziel der Modularisierbarkeit umgesetzt werden konnte.
Abschließend erfolgt eine Übersicht über das entworfene System und es werden potentielle Erweiterungsmöglichkeiten des Projekts erörtert und denkbare Einschränkungen aufgelistet (Kapitel 7.2).

2 Grundlagen

2.1 Informationsverarbeitungsketten

Möchten wir begreifen, was unter dem Begriff Informationsverarbeitungskette (Vgl. Informationsverarbeitungsketten, Kapitel 2.1.8) genau zu verstehen ist, benötigen wir zunächst ein grundlegendes Verständnis vom Kontext einer IVK[1] und davon, von welchen Fachkonzepten eine solche Kette abhängt. Aus diesem Grund werden im Folgenden die relevanten, benötigten Grundbegriffe geklärt.

2.1.1 Information

> „Information ist Information, nicht Stoff, nicht Energie. Ein Materialismus, der dies nicht berücksichtigt, kann heute nicht lebensfähig sein.“ [21, S. 16]

Jenes Zitat ist eine angebrachte Einführung in die versuchte Beantwortung der Frage, was der Begriff Information eigentlich darstellt. Information ist nichts Greifbares, sondern völlig eigens zu behandeln. Wenn wir den Begriff Information beschreiben wollen, kommen wir nicht umhin, auch die Übertragung von Information über einen Kanal zu berücksichtigen:

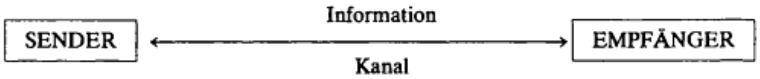


Abbildung 1: Austausch von Informationen

Auf der Suche nach einer allgemeingültigen Definition von Informationen lassen sich etliche Versionen und Abwandlungen finden. Aus diesem Grund wird im Folgenden eine Auswahl an klassischen Ansätzen der Definition von Information dargestellt, welche einen guten Überblick über das weitreichende Verständnis des Begriffs Information bietet.
Im Folgenden ein erster Ansatz, welcher sich besonders auf den Neuigkeitswert einer Information für den Empfänger bezieht:

> „Information ist die Teilmenge von Wissen, die von einer bestimmten Person oder Gruppe in einer konkreten Situation benötigt wird und häufig nicht explizit vorhanden ist.“ [29]

[1] Informationsverarbeitungskette

Die folgende Definition bezieht sich hingegen vorrangig auf den Prozess des Vermittelns der Information:

> „Information ist die Verringerung von Ungewissheit aufgrund von fachlichen Informationsprozessen.“ [29]

Abschließend ein Ansatz, welcher beide bisherigen Definitionen gut zusammenfasst:

> „Information kann charakterisiert werden als die schrittweise und wiederholte Beseitigung von Ungewissheit in Problembehandlungs- und -bewältigungsprozessen, in denen die Zuführung externen Wissens (d.h. beim Problemlöser nicht verfügbaren Wissens) notwendig ist.“ [21, S. 20]

Im Hinblick auf das Verständnis von *Informationsverarbeitungsketten* werden jene Definitionen zugrunde gelegt. So verstehen wir die Zuführung von Informationen als externen Aufbau neuen Wissens, welche zumeist proaktiv mittels fachlicher Recherche in Problembewältigungssituationen geschieht.

2.1.2 Ontologie

Wie der Begriff Information (Vgl. Information, Kapitel 2.1.1), ist auch der Begriff Ontologie sehr oft und unterschiedlich definiert worden, je nachdem in welchem Kontext sich bewegt wird.
Hier interessiert der für die Informatik relevante Ansatz.

Speziell in der Informatik stellt sich oft die Frage, auf welche Art und Weise Informationen sinnvoll repräsentiert werden können und wie sie idealerweise übertragen werden sollten. Während sich Menschen typischerweise Informationen zunutze machen können, indem sie eigenständig auf Informationsquellen wie Lexika, Fachliteratur oder andere Menschen mit Expertenwissen zugreifen und mit eigenem Grund- und Fachwissen vereinen können, benötigen Automaten weitaus festere Regeln. Sie benötigen Anweisungen wie Entscheidungen zu treffen sind, wie Aufgaben durchzuführen sind oder wie eine Kommunikation zu führen ist. Diese Anweisungen enthalten zumeist auch umfangreiche Metainformationen[2]. Automaten benötigen daher bereits ausführlich Wissen darüber, wie jene enthaltenen Informationen und deren zugrundeliegenden Begriffe repräsentiert sind, auf welche Art sie zusammenhängen und wie sie interpretiert werden können.[27]
Ausgehend von jener Problematik führte man den Begriff „Ontologie“ ein, dessen bekannteste Definition im Jahre 1993 vom Informatiker und Erfinder Thomas R. Gruber formuliert wurde:

[2] Metainformation sind ihrerseits Informationen, die andere Informationen beschreiben - insbesondere deren Struktur und die Art wie sie zu interpretieren sind

„An ontology is an explicit specification of a conceptualization.“[27, S. 1]

Eine Ontologie ist also eine Menge von Begriffen, deren Beziehungen zueinander und deren Ableitungen voneinander, welche mit Hilfe einer eindeutig definierten Terminologie beschrieben werden.

Zur Anschaulichkeit sei auf die Abbildung 2 verwiesen, welche exemplarisch eine Menge von Forschern, deren Managern und sonstigen Personen mit deren Beziehungen zueinander darstellt. Die Forscher kooperieren miteinander und berichten an den Manager, während eine vom System unabhängige Person keine Beziehungen zu diesem pflegt.

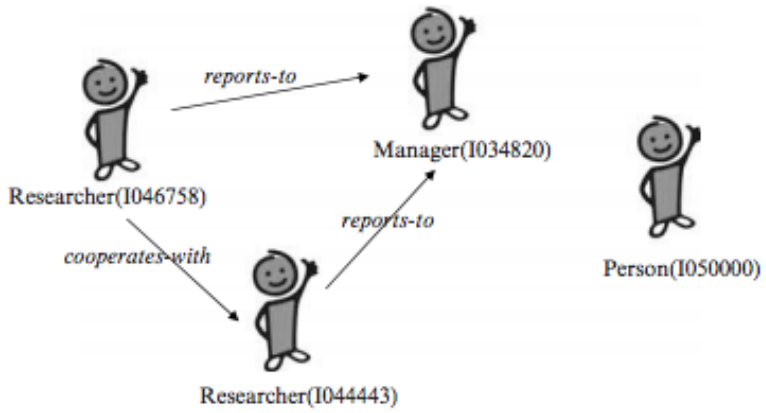


Abbildung 2: Beispielhafte Ontologie [17]

2.1.3 Web

Das Web ist ein über das Internet abrufbares System von Webseiten. Sie werden über die Protokolle HTTP[3] oder HTTPS[4] übertragen und sind via Hyperlinks miteinander verknüpft. Webseiten enthalten Informationen (Vgl. Information, Kapitel 2.1.1), welche in verschiedenen Formaten wie Texten, Grafiken oder Videos abgebildet sind.[2]

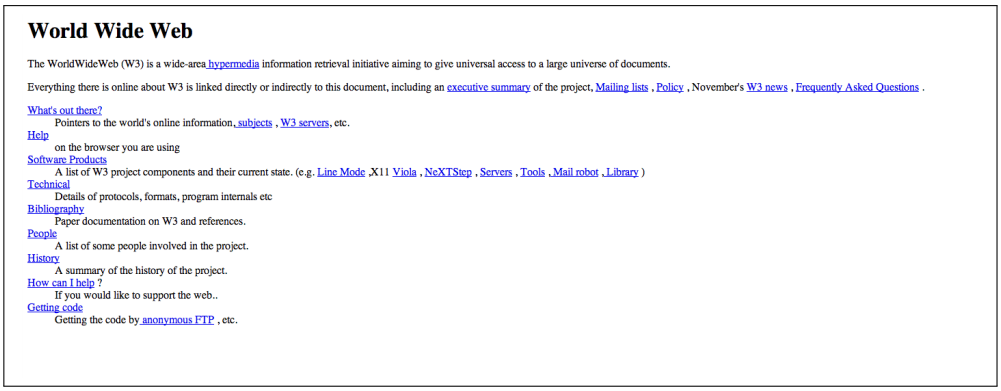
Die Entstehung des Webs ist auf ein Projekt des CERN im Jahre 1989 zurückzuführen, in welchem die beiden Forscher Tim Berners-Lee und Robert Cailliau daran arbeiteten ein Hypertext-System aufzubauen, welches ursprünglich dazu verwendet werden sollte, Forschungsergebnisse innerhalb

[3]Hypertext Transfer Protocol
[4]Hypertext Transfer Protocol Secure

des CERN mit Kollegen auszutauschen. In diesem Zusammenhang entstand folgendes Zitat von Tim Berners-Lee:

> „The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.“[16]

In Abbildung 3 ist eine Ansicht der Rekonstruktion der ersten existierenden Webseite zu sehen.



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

Abbildung 3: Weltweit erste Webseite, welche im Rahmen des CERN-Projekts entstanden ist

Für ein einheitliches Grundkonzept des Web gründete sich am 1. Oktober 1994 am MIT Labratory for Computer Science ein Gremium namens World Wide Web Consortium (W3C), welches es sich zur Aufgabe gemacht hatte, allgemeine Richtlinien und Standards zu entwickeln und festzulegen. [14]

Folgende drei Standards des Web lassen sich als Kernstandards bezeichnen:

- HTTP - Das Protokoll, um über einen Browser Informationen vom Server abzurufen.
- HTML[5] - Die Sprache, in welcher jene Informationen gegliedert und beschrieben werden.
- URL[6] - Die eindeutige Adresse und Lokalisation der Ressource, unter welcher die Informationen zu finden sind.

2.1.4 Semantisches Web

Tim Berners-Lee, jener Begründer des World Wide Web (Vgl. Kapitel 2.1.3), schlug im Jahr 2001 ein Konzept vor, welches die enorm wachsende Menge an Informationen im Web (Vgl. Kapitel 2.1.1) in ihrer Bedeutung auf eine standardisierte Art verknüpfen sollte. Die Notwendigkeit dessen bestand darin, dass im Hinblick auf die enorm schnell wachsende Menge an Informationen im Web eine gezielte maschinelle Verarbeitung der zahlreichen Daten erforderlich wurde. Da für ein angemessenes Verständnis von Informationen und deren Verarbeitung von einer Maschine Ontologien (Vgl. Kapitel 2.1.2) sehr hilfreich sind, entstand jene Erweiterung des Web, genannt Semantisches Web. [32]

Die Idee des Semantischen Web hat Tim Berners-Lee wie folgt formuliert:

> „The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation. The first steps in weaving the Semantic Web into the structure of the existing Web are already under way. In the near future, these developments will usher in significant new functionality as machines become much better able to process and understand the data that they merely display at present.“ [20]

Das Semantische Web baut für die Beschreibung von Informationen auf zahlreiche von der W3C[7] standardisierte Technologien auf. Die für unseren Kon-

[5] Hypertext Markup Language
[6] Uniform Resource Locator
[7] World Wide Web Consortium

text wichtigsten sind URI[8], RDF (Vgl. Kapitel 2.1.5) und OWL (Vgl. Kapitel 2.1.6).[39]
Einen umfangreicheren Einblick in die Technologie-Standards des W3C im Bezug auf das Semantische Web liefert der im Folgenden erkennbare *Semantic Web Stack* (Abbildung 4).

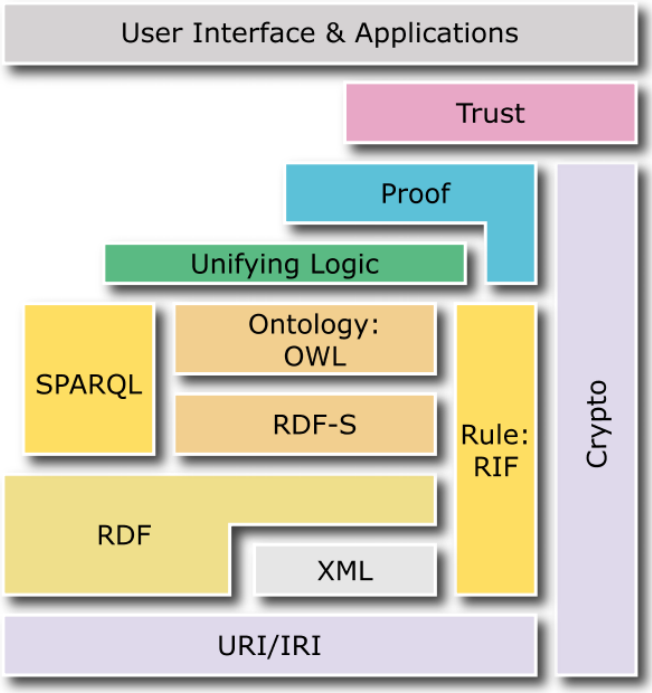


Abbildung 4: Semantic Web Stack[38]

2.1.5 Resource Description Framework

Das Resource Description Framework (RDF) ist ein von der W3C konzipierter Standard zur Beschreibung von Informationen und ihren Metadaten. RDF hat unter anderem den Zweck die verbindende Struktur des Web zu erweitern, indem es zwei Ressourcen und deren Verbindung zueinander beschreibt. Dies wird als ein sogenanntes RDF-Statement oder auch RDF-Tripel bezeichnet.[37, 23]
Ein Tripel enthält stets folgende drei individuelle Einzelteile:[37]

- Das *Subjekt*, also die spezifische Ressource, über welche eine Aussage gemacht wird.
- Das *Prädikat*, welches als eigenständige Ressource eine Eigenschaft des Subjekts darstellt.

[8] Uniform Resource Identifier

- Das *Objekt*, welches dem Wert der Eigenschaft des jeweiligen Subjekts entspricht. Ein Objekt kann eine weitere Ressource oder auch nur ein Literal sein.

Das RDF-Modell ist syntax-neutral in der Darstellung von RDF-Ausdrücken und wird meist als gerichteter Graph[9] dargestellt. In der Anwendung von RDF wird zumeist auf die *Extensible Markup Language* (XML) als primäre Syntax zurückgegriffen, welche mittels der XML-Namespaces eine einfache Möglichkeit bietet, das jeweilige Schema der Ressourcen abzubilden. Ein RDF-Schema (RDFS) beschreibt, ähnlich einem Lexikon, die Bedeutung eines Prädikats näher, um sicherstellen zu können, dass sowohl Autor als auch Leser eines RDF-Statements das gleiche Verständnis über die Beschreibung haben. [37]

Folgend ein beispielhaftes RDF-Statement:[37]

Abbildung 5: RDF-Statement

In dieser Abbildung entspräche die Ressource *http://www.selfm8.com* dem Subjekt. Das Prädikat *Creator* beschreibt das Subjekt näher, indem es ihm das Objekt *Steven Reim* als Literal zur Verfügung stellt. Als Schema wurde die URI *http://description.org/schema/* verwendet. Dieses Tripel würde ausformuliert also etwa heißen:

> *The Creator of the Ressource http://www.selfm8.com is Steven Reim.*

Das RDF/XML-Dokument sähe in diesem Fall aus wie folgt:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
    syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description rdf:about="http://www.selfm8.com/">
    <s:Creator>Steven Reim</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

Abbildung 6: RDF-Statement in XML

[9] Ein Paar (V, E) mit einer Menge V Knoten, welche mittels einer endlichen Menge E gerichteter Kanten (*directed edges*) verbunden sind. Durch eine Kante ist stets ein Anfangsknoten $v \in V$ mit einem Endknoten $w \in V$ verbunden.

2.1.6 Web Ontology Language

Die Web Ontology Language (OWL) ist, wie das Resource Description Framework (Vgl Kapitel 2.1.5), ein vom W3 Consortium eingeführter Standard. Sie ist eine Erweiterung von RDF und bietet zusätzliche Sprachkonstrukte, die erlauben, komplexere Ausdrücke in Prädikatenlogik zu modellieren, für die RDF(S) selbst nicht mächtig genug wäre. Sie entstand aus der Notwendigkeit heraus, ganze Ontologien (Vgl. Ontologie, Kapitel 2.1.2), also für Maschinen verständliche und interpretierbare Ausdrücke, Anweisungen und Informationen abbilden zu können. Sprachkonstrukte, welche komplexe Ausdrucksmöglichkeiten für Informationen enthielten, würden in RDF in der Regel zu sehr hoher Komplexität in der Modellierung und zumeist auch zu einer Unentscheidbarkeit und damit zu einer schlechten Skalierbarkeit führen.[28, 36]
Solche Ausdrücke, welche sich nicht hinreichend in RDF modellieren lassen, sind zum Beispiel:

- „Jede Firma hat stets mindestens einen Angestellten“
- „Der Vorgesetzte meines Vorgesetzten ist auch mein Vorgesetzter“
- „Die Eltern von Heinz Schmidt sind genau Petra Schmidt und Georg Schmidt“

OWL existiert in drei unterschiedlichen Abwandlungen. Sie unterscheiden sich in ihrer Fülle an Funktionen und Komplexität und sollen als Teilsprachen Entscheidbarkeit ermöglichen und die Entwicklung von Tools erleichtern.[28]

- *OWL Full* enthält als einzige Version das gesamte Sprachkonstrukt von RDFS und umfasst OWL Lite sowie OWL DL. Es ist daher sehr mächtig und ausdrucksstark, führt jedoch aufgrund der fehlenden sprachlichen Einschränkungen zur Unentscheidbarkeit der mit OWL Full erstellten Ontologien.
- *OWL DL* (OWL Description Logic) ist Teilsprache von OWL Full und enthält OWL Lite. OWL DL ist in der Lage entscheidbare Ontologien zu modellieren, enthält jedoch dadurch gewisse Enschränkungen in der RDFS-Modellierung.
- *OWL Lite* ist eine Teilsprache von OWL DL und OWL Full und ist daher weniger mächtig und aussagestark, bietet jedoch genau wie OWL DL die Möglichkeit, entscheidbare Ontologien zu modellieren und bietet eine geringere Komplexität und eine vereinfachte Implementierung von Tools.

2.1.7 Linked Open Data

Linked Open Data (LOD) ist ein Sammelbegriff für jene frei verfügbaren Informationen (Vgl. Information, Kapitel 2.1.1) im Web (Vgl. Web, Kapitel 2.1.3), welche mittels URIs beschrieben sind, über HTTP abgerufen werden können, auf Standards wie RDF (Vgl. Resource Description Framework, Kapitel 2.1.5) oder OWL (Vgl. Web Ontology Language, Kapitel 2.1.6) für die Modellierung aufbauen, sowie zusätzlich per URI auf weitere Ressourcen verweisen. Diese Beschreibung beruht auf einem von Tim Berners-Lee im Jahre 2006 entworfenen Konzept für Linked Data und definiert genau jene vier Regeln:[35, 34]

1. „Use URIs as names for things“
2. „Use HTTP URIs so that people can look up those names.“
3. „When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL[10])“
4. „Include links to other URIs, so that they can discover more things“

Das Konzept der Linked Open Data baut auf genau diesen vier Regeln auf und erweitert sie um die Bedingung, dass es sich stets um frei verfügbare Daten und Informationen handeln muss.
Folgende Definition entstand hieraus:

> „Offene vernetzte Daten sind sämtliche Datenbestände, die im Interesse der Allgemeinheit der Gesellschaft ohne jedwege Einschränkung zur freien Nutzung, zur Weiterverbreitung und zur freien Weiterverwendung frei zugänglich gemacht und über das World Wide Web miteinander vernetzt sind.“[34]

2.1.8 Informationsverarbeitungsketten

In den vorangegangenen Kapiteln 2.1.4 und 2.1.7 wurde aufgezeigt, wie Informationen (Vgl. Kapitel 2.1.1) semantisch verknüpft werden können um Zusammenhänge abbilden und Ontologien (Vgl. Kapitel 2.1.2) schaffen zu können. Zugängliche Informationen können jederzeit aufgegriffen werden, um mit eigenen Informationen verknüpft zu werden und dadurch neuartige Informationen zu schaffen. Mittels Standards wie RDF (Vgl. Kapitel 2.1.5) und OWL (Vgl. Kapitel 2.1.6) sind wir in der Lage, unsere Informationen und Ressourcen so zu kennzeichnen, dass sie für Maschinen verständlich, auffindbar und verarbeitbar sind.
Dieser gesamte Ablauf, der entsteht, wenn Informationen einer beliebigen Quelle an einer anderen Stelle aufgegriffen, gefiltert und weiterverarbeitet

[10] *SPARQL* ist eine Abfragesprache für RDF-Dokumente.

werden, nennen wir *Informationsverarbeitungskette*.

Eine Informationsverarbeitungskette besteht grundsätzlich aus einer *Quelle* und einer *Senke*. Zusätzlich ist eine Verbindung beider Komponenten vonnöten.
Die **Quelle** ist der Ursprung einer Information auf die sich bezogen wird. Dies kann beispielsweise eine Datenbank sein, welche uns die benötigte Information mittels eines speziellen SQL-Statements preisgibt, dies kann aber unter anderen ebenso eine spezielle Excel-Datei, ein Blog-Eintrag, ein DBpedia-Dokument[11] sein. Eine Quellkomponente benötigt daher Informationen darüber, mit welchen Mitteln respektive auf welche Art und Weise sie die benötigte Information abzurufen hat.
Eine **Senke** ist das jeweilige Ziel, wo die aufbereitete Information persistiert oder platziert wird. Dies können ebenso beliebige Medien wie eine weitere Datenbank, Excel-Datei oder die eigene Webseite sein. Handelt es sich bei jener Komponente nicht um das endgültige Ziel, sondern ist diese nur Zwischenstation des bekannten Flusses jener Information, heißt diese Komponente *Transformation* und ist für den nächsten Schritt der Informationsverabeitung erneut Informationsquelle. Eine Senkenkomponente benötigt dementsprechend Informationen darüber, wie sie die erhaltenen Daten korrekt verarbeitet, manipuliert und gegebenenfalls persistiert.
Die **Verbindung** von Quelle zu Senke stellt den Informationsfluss dar.

2.2 Google Web Toolkit

2.2.1 Eigenschaften

Das Google Web Toolkit (GWT) ist ein von Google im Jahr 2006 veröffentlichtes Werkzeug zur Erstellung von *Rich Internet Applications*[12] im Web. Es hat den Nutzen, umfangreiche und komplexe JavaScript-Anwendungen in Java zu entwickeln. Die erstellte Anwendung existiert daher auf Clientseite und wird direkt im Browser ausgeführt. Mittels eines eigenen integrierten Compilers wird der entworfene Java-Code zur Kompilierzeit in JavaScript übersetzt und kann dadurch als reines HTML-JavaScript-Kombinat auf einem beliebigen Webserver deployt werden. GWT bietet Kompilierunterstützung für einen Großteil der Standard-API der Java-Plattform. Das GWT SDK bietet zudem ein umfangreiches Eclipse-Plugin und einen integrierten Application Server, mit welchen, zusammen mit der Benutzung eines GWT-Browser-Plugins, die Anwendung während der Entwicklung bereits

[11] DBpedia stellt sämtliche Artikel der deutschen Wikipedia strukturiert mithilfe von RDF zur Verfügung, damit sie automatisiert über die Abfragesprache SPARQL abgerufen werden können.[3]

[12] Rich Internet Applications (RIA) sind Anwendungen, die eine besonders reichhaltige Menge an Interaktionsmöglichkeiten, wie beispielsweise Drag and Drop, 3D-Effekte oder Animationen, bieten.

im Browser begutachtet werden kann. Der Application Server unterstützt dazu das *Hot Deployment*, wodurch getätigte Code-Änderungen auch ohne Server-Neustart eingesetzt werden.
Folgende Fähigkeiten des Google Web Toolkits sind besonders hervorzuheben:

Remote Procedure Calls (RPC), welche die Kommunikation mit dem Server ermöglichen. Clientseitig wird hierzu ein einfaches Java-Interface erstellt, welches mit geringem Aufwand von einer auf dem Server existenten Klasse implementiert werden kann. Solche in reinem Java entwickelten Server-Requests werden zu gewöhnlichen AJAX-Requests übersetzt, welche mittels SOAP die jeweils aufgerufene Methode mit ihren Parametern aufrufen. Sie sind daher asynchron und blockieren nicht das User Interface. Der Client implementiert für eine angemessene Verarbeitung der irgendwann ankommenden Response die zwei Methoden *onSuccess(...)* und *onFailure(...)*.

Java Debugging zur Entwicklungszeit. Dies bedeutet, dass GWT im Debug Mode zwar JavaScript-Komponenten auf dem Application Server platziert und vom Browser ausführen lässt, jedoch das Debugging des Codes direkt im Java-Code in Eclipse ermöglicht. So können wie gewohnt Break Points gesetzt werden und Zeile für Zeile durch den Code geschritten werden. Zusätzlich können indes Variablen mit ihren potentiell vorhandenen Subelementen inspiziert werden und der Stack Frame angehaltener Threads angezeigt werden.[8]

Das Google Web Toolkit bietet ein umfangreiches Repertoire an **Widgets und Panels**.
Widgets sind vordefinierte HTML-Komponenten, welche Unterstützung für verschiedene JavaScript-Funktionen bieten. So bieten Widgets (bspw. das Button-Element) in GWT Modifikatoren für sämtliche in HTML verfügbare Attribute und sonstigen Anpassungen mittels Methoden wie *setName(...)* oder *setEnabled(...)*.
Panels hingegen entsprechen ganzen HTML-Konstrukten, welche zum Layouting ihrer Kind-Elemente genutzt werden können. GWT bietet Panels wie das *HorizontalPanel*, in welchem Subkomponenten horizontal nacheinander angeordnet werden, oder das DockLayoutPanel, in welchem die Kinder nach Himmelsrichtungen (*panel.addNorth(...), ...*) angeordnet werden können. Zusätzlich bietet es auch Panels, welche TabBars und Accordions aufbauen können.

Nachdem über lange Zeit in GWT das GUI[13] der Applikationen lediglich programmatisch aufgebaut werden konnte, hat Google mit GWT 2.0 auch

[13] Graphical User Interface

einen deklarativen Ansatz eingeführt, den **UiBinder**. Seitdem können Komponenten eigene UI-Files besitzen, in welchen in XML-Struktur beliebige Elemente deklariert und hinzugefügt werden können. Der Aufbau eines Panels mit Label und Textfeld sähe in Java-Form folgendermaßen aus:

```
VerticalPanel panel = new VerticalPanel();
Label lbl = new Label("Name:");
TextBox box = new TextBox();
box.setPlaceholder("Bitte eingeben...");
box.setMaxLength(100);
panel.add(lbl);
panel.add(box);
```

Abbildung 7: Panel in GWT - Java

Deklariert im UiBinder würde jenes Konstrukt hingegen folgendermaßen aufgebaut sein:

```
<g:VerticalPanel>
  <g:Label text="Name:" />
  <g:TextBox placeholder="Bitte eingeben..."
    maxLength="100" />
</g:VerticalPanel>
```

Abbildung 8: Panel in GWT - UiBinder

So umfangreich GWT in der aktuellen Version 2.6 (Stand Februar 2014) auch ist, irgendwann stößt es auch an Grenzen dessen, was in Java nach JavaScript übersetzt werden kann. Aus diesem Grund bietet GWT auch die Möglichkeit, puren JavaScript-Code direkt innerhalb der Java-Klassen zu schreiben, genannt **JavaScript Native Interface** (JSNI). Dies funktioniert, indem das JavaScript direkt innerhalb von eigenen als *native* deklarierten Java-Methoden platziert wird und dann wie gewohnt beliebig von anderorts aufgerufen werden kann. Diese Methoden können bekanntermaßen Parameter annehmen. Das JavaScript ist zusätzlich auch in der Lage, direkt auf Java-Methoden und -Felder zugreifen zu können. Beim Debuggen kann so innerhalb der Java-Klassen sowohl durch den Java-Code als auch direkt durch den JavaScript-Code gelaufen werden.
Ein Beispiel für eine solche JSNI-Methode ist in Abbildung 9 zu sehen:

```
private static native void example(boolean sayHi)/*-{
  if (sayHi) {
    var name = this.@de.fub.agse.meq.mangold.client.
        Mangold::userName;
    $wnd.alert('Hello, ' + name);
  }
}-*/
```

Abbildung 9: Beispiel einer JSNI-Methode

Ebenso stellt GWT, ähnlich wie auch andere UI-Frameworks in Java, ein **Event-Handling** zur Verfügung. Es ist in der Lage allgemeine JavaScript-Events wie *onChange(...)* oder *onSubmit(...)* in Java abzubilden. Ein Beispiel, wie dies umgesetzt wird, ist in Abbildung 10 zu erkennen, in welcher eine JavaScript-Meldung aufgerufen wird, sobald ein Button geklickt wird.

```
Button button = new Button("Click me");
rootPanel.add(button);
button.addClickHandler(new ClickHandler() {
  public void onClick(ClickEvent event) {
    Window.alert("Great! You just clicked me!");
  }
});
```

Abbildung 10: Beispiel eines GWT-Button-Events

Zusätzlich lassen sich mit GWT auch eigens definierte Events erstellen. Dies ist zum Beispiel nützlich, wenn eine ausgeführte Aktion in einer Unterkomponente (bspw. innerhalb eines Modal-Panels) diverse Aktionen in einer übergeordneten Komponente hervorrufen soll. Dafür wird ein sogenannter EventBus aufgebaut, welcher in der Lage ist, Events zu transportieren. Anschließend müssen sowohl ein neues Event, welches gefeuert werden kann, als auch ein separater Handler entworfen werden, welcher mit diesem Event umzugehen weiß. Beides vereint ist in Abbildung 11 zu sehen.

```java
public class AddComponentEvent extends Event<
    AddComponentEvent.Handler> {

  public interface Handler {
    void onAddComponent(AddComponentEvent event);
  }

  private final Component component;

  public AddComponentEvent() {
    super();
    component = null;
  }

  public AddComponentEvent(Component component) {
    super();
    this.component = component;
  }

  public static HandlerRegistration register(EventBus
      eventBus,
       AddComponentEvent.Handler handler) {
    return eventBus.addHandler(TYPE, handler);
  }

  private static final Type<AddComponentEvent.Handler>
       TYPE = new Type<AddComponentEvent.Handler>();

  @Override
  public Type<AddComponentEvent.Handler>
      getAssociatedType() {
    return TYPE;
  }

  @Override
  protected void dispatch(AddComponentEvent.Handler
      handler) {
    handler.onAddComponent(this);
  }

  public Component getComponent() {
    return this.component;
  }
}
```

Abbildung 11: Ein Event in GWT

Sodann kann folgendermaßen jenes Event gefeuert werden:

```
ClientModel.EVENT_BUS.fireEvent(new
    AddNewComponentEvent(component));
```

Abbildung 12: Feuern eines Events in GWT

An einer oder mehreren anderen Instanzen und Klassen kann nun auf jenes Event gelauscht werden, indem der entworfene Handler für dieses Event auf dem genutzten EventBus registriert wird:

```
AddComponentEvent.register(ClientModel.EVENT_BUS,
  new AddComponentEvent.Handler() {
    public void onAddComponent(AddComponentEvent event
        ) {
      // Do sth.
    }
});
```

Abbildung 13: Registrieren eines Events in GWT

2.2.2 Statistiken

Seit Einführung erfreut sich das GWT einer immer weiter steigenden Beliebtheit. Da es mittlerweile ein sehr etabliertes Framework ist und die Anzahl an nutzbaren, unterstützenden Bibliotheken und Wrapper immer weiter steigt, entstehen im Web auch zunehmend umfangreiche Projekte, welche auf GWT aufbauen. Beispiele hierfür sind zum Beispiel das im Jahre 2011 auf das GWT portierte *studiVZ*[14], *Google Groups*[15] und *Google Adwords*[16][5], sowie die Web-Portierung des erfolgreichen Smartphone-Spiels *Angry Birds*[17][6].

Nichtsdestotrotz ist der tatsächliche Anteil an GWT-Anwendungen unter den größten Webseiten noch sehr gering, wie in Abbildung 14 zu sehen ist. So erstreckt sich die Quote von GWT-Anwendungen unter den bedeutendsten 10.000 Webseiten auf gerade einmal 0,25%; bei den eine Million bedeutendsten Webseiten liegt die Quote sogar lediglich bei 0,02%.

[14] *http://www.studivz.net/*
[15] *https://groups.google.com*
[16] *https://adwords.google.de*
[17] *http://chrome.angrybirds.com/*

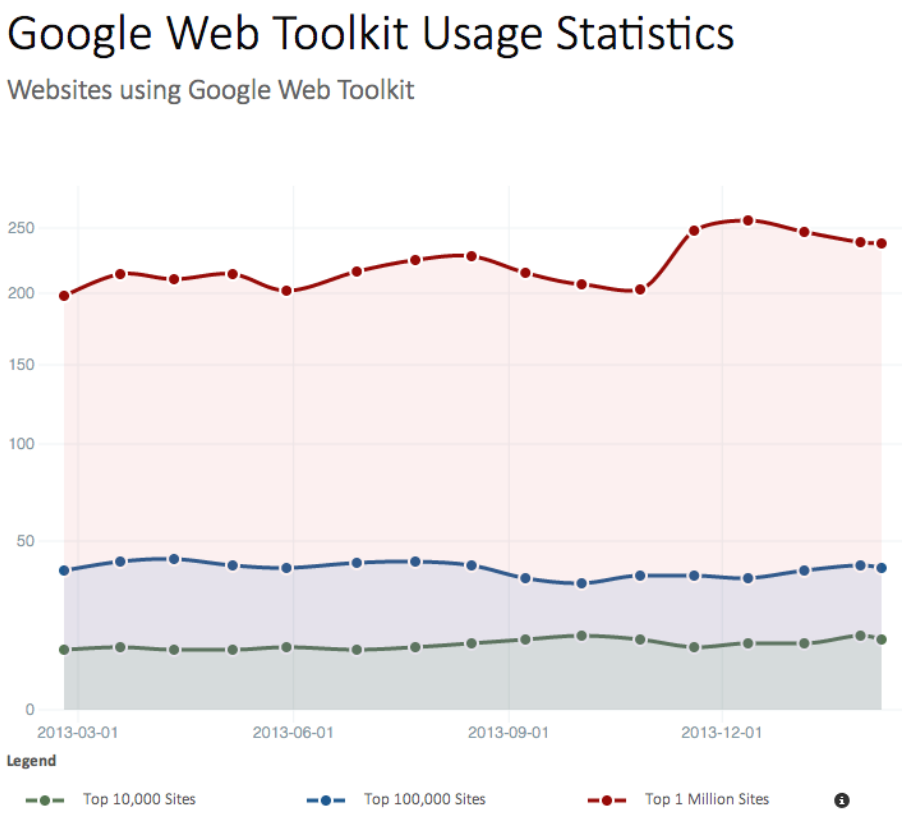


Abbildung 14: Google Web Toolkit - Usage Statistics [7]

Zudem ist hervorzuheben, dass die Häufigkeit des Einsatzes von GWT als Basis-UI-Technologie sehr vom jeweiligen Kontext der entwickelten Plattform abhängt. So zeigt sich, wie in Abbildung 15 zu sehen, dass insbesondere im Business-Bereich in Erwägung gezogen wird, auf GWT aufzubauen, während sich besonders für Chat-Plattformen und komplexe Software-Systeme ein Einsatz von GWT nicht besonders anzubieten scheint.

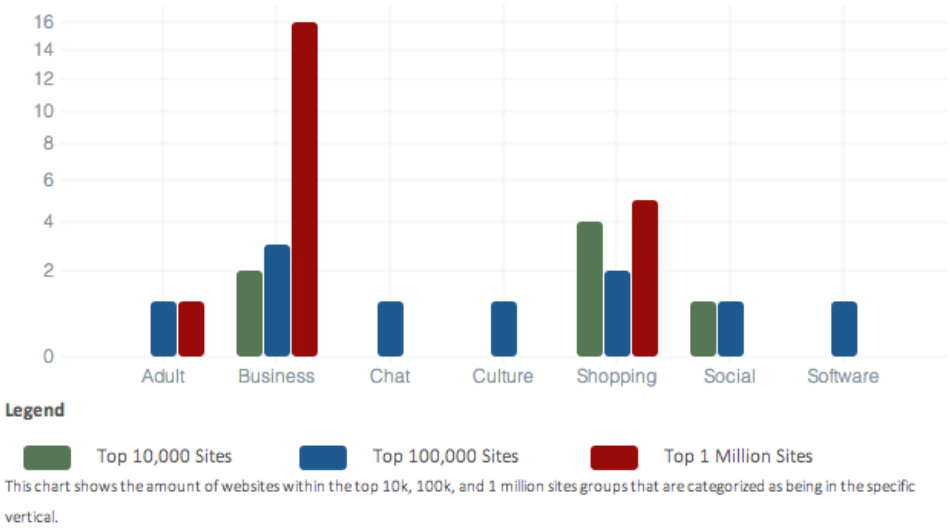


Abbildung 15: Google Web Toolkit - Verteilung auf Themenbereiche [7]

Solche Anwender, die erfolgreich Applikationen mit GWT entwickelt haben, wurden einst von der Software-Firma *Vaadin*[18], dessen Kernprodukt ein auf GWT aufbauendes Framework ist, gefragt, welches ihrer Meinung nach die wichtigsten Vorteile von GWT seien [13]. Die mit Abstand häufigste Antwort war die Befreiung von der für die meisten Webentwickler größten Last in der Webentwicklung: die Browserabhängigkeit. Mit GWT müssen sich Entwickler kaum mehr Gedanken um die vom Nutzer eingesetzte Umgebung machen, sondern können befreit davon abstrahiert ihren Code schreiben, während browserbedingte Abwägungen von GWT selbst durchgeführt werden. Weitere Antworten lassen sich auf Abbildung 16 finden.

[18] *https://vaadin.com*

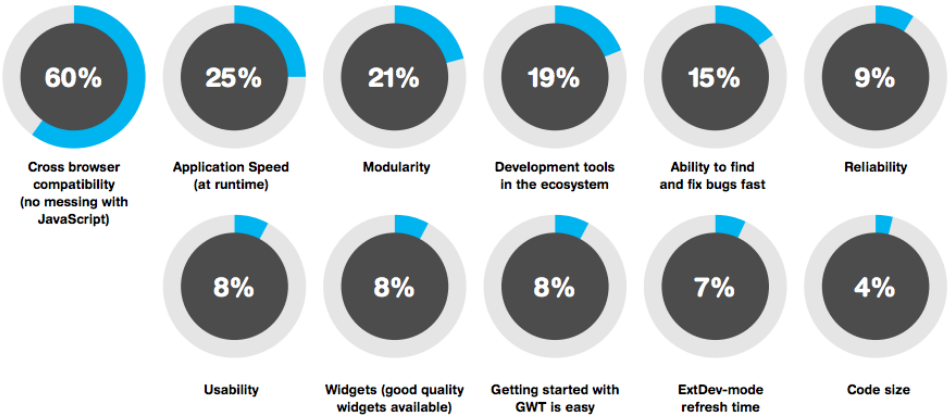


Abbildung 16: Google Web Toolkit - Benefits [13]

3 Aggregation von Informationen - Eine Recherche

Aufgrund der immer größer werdenden Flut an Informationen, die von Personen und Systemen aufgenommen respektive verarbeitet werden müssen, ist die Idee, Informationen verschiedener Ursprungsorte zu verknüpfen und auf eigene Art zu verarbeiten oder darzustellen, nicht neu. Tatsächlich versuchten bereits mehrere große Unternehmen, größtmöglich generische und leicht bedienbare Web-Applikationen zu entwerfen, mit denen übersichtliche Informationsverarbeitungsketten kreiert werden können. Viele dieser sogenannten Mashup-Technologien scheiterten jedoch bereits während der Entwicklung an der Vielschichtigkeit der zu berücksichtigen Arten von Informationsquellen und auch an ihren eigenen Ansprüchen, diese Vielschichtigkeit für jeden Nutzer besonders simpel bedienbar und verständlich machen zu wollen. Beispiele für Systeme, die nicht über die Entwicklungsphase hinausgekommen sind, sind etwa das Projekt **Popfly** von Microsoft oder der **Google Mashup Editor** von Google.

Popfly setzte auf die Microsoft-eigene *Silverlight*-Technologie[19] und sollte dem Nutzer in einer aufwendig gestalteten Grafikoberfläche die unkomplizierte Möglichkeit bieten, Mashup-Darstellungen oder auch eigene Webseiten und Programmcodes entwerfen zu können. Es enthielt sogar einen sogenannten *Game Creator*, welcher Nutzern eigene Spiele erstellen oder bestehende Spiele erweitern lassen sollte, um sie anschließend auf *Facebook* exportieren zu können oder als *Windows Live Gadget* zu nutzen.
Das Projekt wurde am 16. Juli 2009 eingestellt. Als Grund gab Microsoft die momentane wirtschaftliche Lage und den harten, sich selbst auferlegten, Sparkurs an.[11]

Der *Google Mashup Editor* verstand sich mehr als reinen Editor statt als aufwendig gestaltete Oberfläche. Der Anwender fand bei Aufruf einen einfachen Text-Editor vor, welcher, mit einem zusätzlichen Feed-Browser und einer sogenannten *Sandbox* versehen, ihn verschiedene Feed-Inhalte mittels JavaScript, XML, HTML, CSS und Google-eigener GME-Tags aufgreifen und in einer eigenen Oberfläche darstellen ließ.[4]
Am 15. Januar 2009 wurde auch dieses Projekt eingestellt. Es wurde jedoch in die *Google AppEngine* überführt.

Ein Projekt, welches auch heute noch weiterbesteht, ist **Yahoo Pipes**[20].
Pipes ist ein sehr umfangreiches Tool, welches dem Nutzer auf übersichtliche

[19] Ein Framework zum Erstellen von *Rich Internet Applications*, welches nah an Adobe Flash angelehnt ist
[20] http://pipes.yahoo.com/

Weise eine grafisch dargestellte Oberfläche anbietet, in welcher er sich sein gewünschtes Mashup aufbauen kann. Er zieht dafür mittels Drag&Drop aus der *Library* das gewünschte Element in das sogenannte *Canvas*, welches den eigentlichen Arbeitsplatz darstellt. Innerhalb des Canvas kann der Nutzer die Parameter der Komponenten ausfüllen und anpassen und Verbindungen zwischen zwei Elementen erstellen. Es bietet zahlreiche Komponenten in der Library, welche für die Verarbeitung genutzt werden können. Mögliche Quellen sind etwa eine CSV-Datei, ein Feed, Flickr-Bilder oder eine Webseite. Zudem kann der Nutzer diverse Tools verwenden, um etwa Ausgabestrings einer Quelle zu manipulieren, Datenmengen zu sortieren oder teilen und vereinen, oder Geo-Daten zu erhalten. Diese Daten werden dann am Ende dieser sogenannten *Pipe* in das *Pipe Output* gegeben. Eine Beispieldarstellung einer Pipe ist in Abbildung 17 zu sehen.

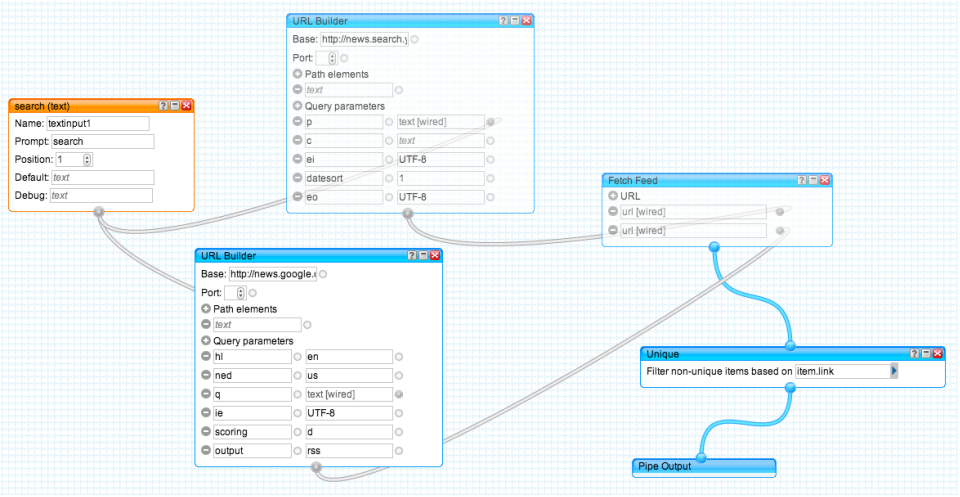


Abbildung 17: Beispielhafte Pipe in Yahoo Pipes

Im abgebildeten Beispiel wird dem Anwender die Möglichkeit gegeben, einen Suchstring einzugeben, welcher innerhalb der News Feeds von Yahoo und Google gesucht wird. Dafür wird, wie in der Abbildung 17 links zu erkennen, ein *TextInput*-Element in das Canvas eingebunden. Dieses ist mit zwei *URL Buildern* verknüpft, welche die URL für die News-Feeds aufbauen. Auf diese beiden URLs bezieht sich das *Fetch Feed*-Element, welches dann, nachdem aus dem Ergebnis noch doppelte Einträge entfernt wurden, die Daten in das *Pipe Output* gibt.

Der Nutzer kann, sobald eine fertige Pipe entworfen wurde, mittels des Links *Run Pipe* diese Pipe ausführen. Er wird dann auf eine externe Seite geführt, auf der er, wie im aktuellen Beispiel, einen Such-String eingibt und auf einen

weiteren Button *Run Pipe* klickt. Ein beispielhaftes Ergebnis dieser Suche wird in Abbildung 18 dargestellt.

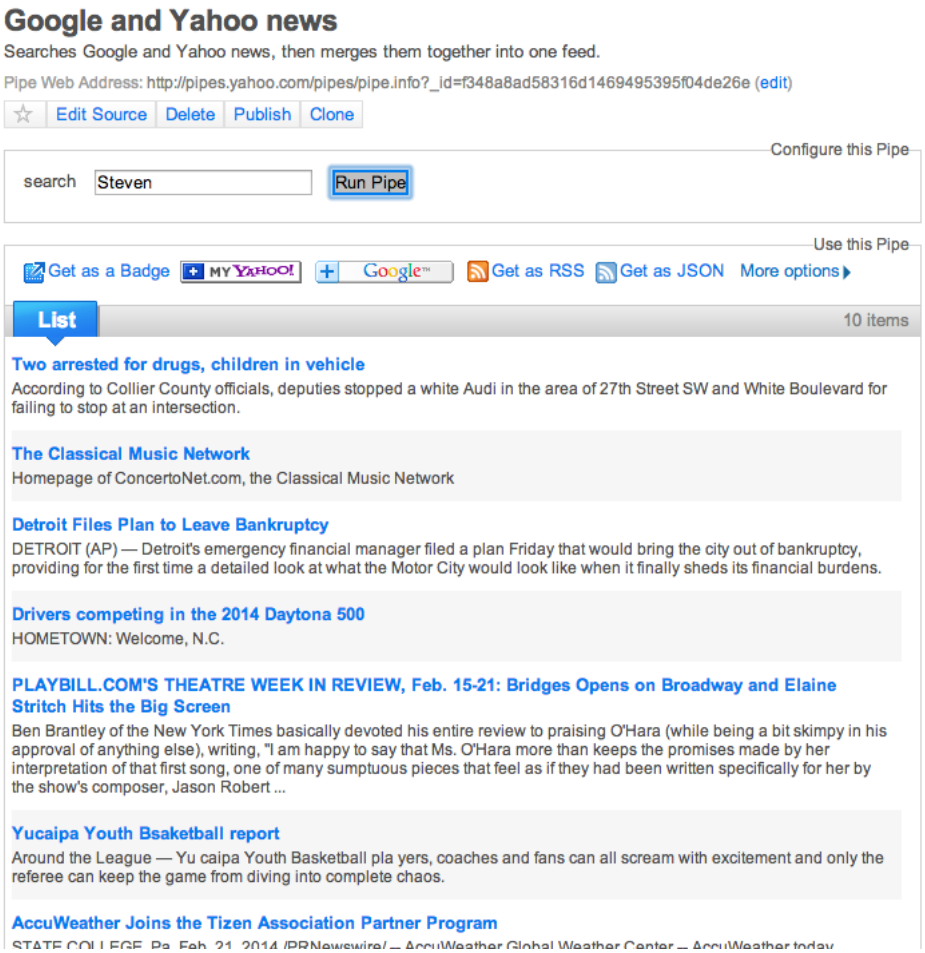


Abbildung 18: Beispielhaftes Ergebnis einer Pipe

Yahoo Pipes ist ein sehr umfangreiches System. Für unsere Ansprüche ist es jedoch leider noch zu unflexibel, da sowohl die Menge an verfügbaren Quellen nicht umfangreich genug ist und sich keine Möglichkeit bietet, neue Elemente zur *Library* hinzuzufügen. So fehlen zum Beispiel die Möglichkeiten, sich mit existierenden Datenbanken direkt zu verbinden oder einen SPARQL-Endpunkt anzubinden. Zudem kann der Anwender innerhalb der Pipe nicht entscheiden, was am Ende der Kette mit diesen fertigen Daten geschehen soll. Er ist auf die Ausgabe auf der externen Seite beschränkt und müsste eventuelle Weiterverarbeitungen, wie Persistieren in einer neuen Datenbank, manuell durchführen, indem er sich das Pipe-Ergebnis etwa als JSON exportiert und dann in die Datenbank importiert.

Nichtsdestotrotz lassen sich zweifellos auch Parallelen zwischen *Yahoo Pipes* und Mangold finden. Speziell an der Nutzeroberfläche von Pipes wurde sich entfernt angelehnt. So bietet auch Mangold eine übersichtliche Art Komponenten im Editor darzustellen, welche durch Linien (*Connections*) miteinander verknüpft sind und über eine Toolbar um weitere in Gruppen dargestellte Komponenten erweitert werden können. Aus Gründen des geringen Platzes auf dem Bildschirm kleiner Notebooks und eines potentiell sehr umfangreich groß und komplex gestalteten Workspaces entscheide ich mich jedoch für eine noch kompaktere Darstellung der Toolbar, welche über einen Klick in einer zentralen, minimal gehaltenen MenuBar erreichbar sein soll. Ebenso sind die diversen Komponenten im Editor von Mangold deutlich kompakter dargestellt und zeigen auf den ersten Blick lediglich den Namen der Komponente auf. Auf Eigenschaften und Verhaltensweisen der Komponenten lassen sich erst über eine separate Auswahl in der MenuBar zugreifen. Der Ansatz von *Yahoo Pipes* ist dementsprechend einfacher zugänglich und bietet eine sehr übersichtliche Möglichkeit, neue Komponenten der Arbeitsfläche hinzuzufügen und vorhandene Komponenten zu bearbeiten und zu erweitern. Jedoch tendiert es aufgrund der Größe der dargestellten Komponenten sehr schnell dazu, unübersichtlich zu werden und sorgt dafür, dass der Nutzer sehr oft im Editor scrollen muss, um die gewünschte Stelle zu finden, wodurch dann andere Komponenten aus dem Fokus geraten.

4 Konzeptionierung

4.1 Anforderungen

Eine erfolgreiche Implementierung des Prototypen, im Folgenden *Mangold* als Projektname genannt, erfordert zunächst eine detaillierte Erhebung der Anforderungen.
Zu entwickeln ist ein webbasierter Editor, welcher mittels Drag&Drop einzelne Komponenten einer umfangreichen Informationsverarbeitungskette darstellen kann und in der Lage ist, sämtliche Möglichkeiten von Informationsquellen und -senken abzubilden. Dieser Editor soll nebst der Funktion der übersichtlichen Nachbildung von existierenden Informationsverarbeitungsketten zudem auch die Funktion einer Erstellung von neuartigen Informationsverarbeitungsketten bieten, so dass ein Nutzer durch einfach zu bedienende Hilfsmittel die Möglichkeit erhält, sich komfortabel selbst Informationen von diversen Orten zu besorgen und sie konkateniert und modifiziert in eigenen Senken platziert. Auf eine nähere Definition des Begriffs *Information* wurde bereits in Abschnitt 2.1.1 näher eingegangen.

Hierzu sollen die verwendeten Komponenten umfangreich und generisch Informationen enthalten können, wie deren anzuwendende Prozedur auszusehen hat, welche Art von Informationsquelle, bzw. -senke sie darstellen und bedienen, sowie welche Informationen und Parameter sie für ihren Prozessablauf dynamisch angegeben benötigen.
Verbunde von Komponenten sollen durch Linien visualisiert werden, welche anhand der Verbindungspunkte jeder einzelnen Komponente ihre Informationen übertragen sollen. Ein Verbindungspunkt spiegelt den Übergang der Komponente in den generischen Informationsfluss, bzw. umgekehrt, wider. So stellt der Verbindungspunkt einer Quelle üblicherweise eine gesonderte Abfrage der Informationen der Komponente dar, während der Verbindungspunkt einer Senke die Informationen des Flusses so formt, dass die Komponente mit diesen erhaltenen Informationen umzugehen weiß, bzw. die Informationen korrekt platziert oder persistiert.
Eine Komponente soll generell über Informationen zu ihrem Prozessablauf verfügen. Diese Informationen sollen der Komponente bereits bei ihrem Einfügen in den Editor bekannt sein und in Form eines Programmcodes, folgend *Prozess-Code* genannt, vorliegen und zur korrekten Verarbeitung nur noch diverse Eingaben an Parametern benötigen. Soll zum Beispiel eine generische Komponentenquelle geschaffen werden, welche Informationen von einer MySQL-Datenbank besorgen soll, dann soll diese Komponente bereits über das Wissen verfügen, wie sie sich zu einer beliebigen MySQL-Datenbank zu verbinden hat, also welchen JDBC-Treiber sie benötigt und welche Prozesse aufgerufen werden müssen, um sich zu der gegebenen Datenbank zu verbinden und Daten aus dieser Verbindung zu besorgen. Sie muss zudem

wissen, welche zusätzlichen, variierenden Parameter sie benötigt, um generell Informationen aus dieser Quelle zu beziehen. Diese Parameter sollen dann bei Nutzung der Komponente im Editor vom Nutzer selbst angegeben werden. Im Beispiel der MySQL-Datenbank wären solche Parameter typischerweise Host, Port und Name der Datenbank, sowie der Nutzername und das Passwort für den Zugriff auf die Datenbank. Im jeweiligen Verbindungspunkt wird dann zudem der Parameter für das benötigte SQL-Statement bereitgestellt, mit welchem die für diese Verbindung eigentlich benötigten Informationen bezogen werden. Der Prozess-Code der Komponente greift dieses Statement dann auf, wendet es an und vermittelt das Ergebnis weiter.

Veränderungen am Editor müssen Rückgängig gemacht werden können. Bei Veränderungen der Komponenteneigenschaften, Hinzufügen oder Löschen von Komponenten und Verbindungen von Komponenten und auch bei Verschiebungen von Komponenten muss dem Nutzer die Möglichkeit gegeben werden, seine Änderungen zu widerrufen oder gegebenenfalls widerrufene Änderungen auch wiederholen zu können. Diese Möglichkeit soll der Nutzer erhalten, indem er im Kontextmenü der Menüleiste die zwei Links **Rückgängig** und **Wiederholen** auswählen kann.

Zudem soll der Editor ein umfangreiches Persistenzmodell beinhalten.
Einerseits ist das Ziel der Umsetzung des Prototypen, eine intelligente Arbeitsplatz-Verwaltung (künftig Workspace-Verwaltung genannt) zu schaffen. Er soll also eine Möglichkeit bieten, verschiedene, in ihrer Komplexität variierende, Projekte von abgebildeteten Informationsverarbeitungsketten bauen und diese bei Bedarf laden und speichern zu können. Dieses System ist insofern intelligent, als dass es jederzeit im Hintergrund Veränderungen (Deltas) im Editor registriert und diese dynamisch vom Nutzer unbemerkt persistiert.
Ebenso soll der Prototyp ein einfaches User-Management-System haben, mittels welchem sich der Nutzer am System anmelden kann und auf seine persönlichen Informationsverarbeitungsmodelle zugreifen kann. Hierzu muss eine Anmeldeoberfläche entstehen, welche die Anmeldedaten des Nutzers auf Korrektheit zu überprüfen hat und ihm gegebenenfalls Zugriff zur Oberfläche gestattet. Bei falschen Anmeldedaten soll dementsprechend eine Fehlermeldung erscheinen.

Der Prototyp soll ebenso eine umfangreiche Administrationsoberfläche erhalten, in welchem existierende Komponenten bearbeitet und gelöscht, sowie neue Komponenten flexibel hinzugefügt werden können. Hier soll eingestellt werden können, was der genaue Arbeitsablauf der Komponente zu sein hat und welche zusätzlichen Parameter diese Komponente benötigt. Es soll ein separater Editor-Bereich geschaffen werden, in welchem der Prozess in Form von Groovy- oder Java-Code entwickelt wird. Das Beispiel der MySQL-Datenbank-Quelle aufgreifend, würde dieser Code also den Im-

port der JDBC-Bibliothek darstellen, den Treiber laden, mithilfe dessen eine Verbindung aufbauen und potentiell auftretende Exceptions abfangen, bzw. mit diesen umzugehen wissen. Insbesondere soll der Code dann das gegebene SQL-Statement verwenden, um die Daten zu besorgen und zurückzugeben.

Die Anforderungen an den Editor selbst sind verstärkt kosmetischer Natur und sollen für eine komfortable Bedienung sorgen. So sollen die Komponenten im Editor verschiebbar (Drag&Drop) sein, damit der Nutzer sie jederzeit neu anordnen kann. Ebenso soll der Editor zoombar, also in der Größe seines abgebildeten Bereichs variabel, sein können. Zusätzlich soll der Nutzer entlang des angezeigten Bereichs des Editors navigieren können, also im Editor beliebig nach oben, unten, links und rechts springen können.

4.2 Vorbedingungen

Das Projekt wird auf einer Grundlage von Dr. Edzard Höfig basieren. Er entwickelte im Voraus bereits ein System, welches einen Editor mit Drag&Drop-Funktionalität darstellen und einzelne Komponenten des Editors über eine Seiten-Navigationsleiste darstellen konnte.
Dementsprechend bestand auch bereits eine Architektur für die Verwendung von Quell- und Senkenkomponenten und deren Verbindungen zueinander, welche umfangreich wiederverwendet werden kann. Sie bietet bereits eine funktionierende Editor-Ansicht, welche mithilfe des *Canvas*-Elements von HTML5 und *kinetic.js* Komponenten innerhalb des Canvas rendert und diese via Drag&Drop auch bewegen kann. Zudem sind Komponenten und Verbindungspunkte bereits in der Lage, vorher im Code festgelegte Eigenschaften zu enthalten und diese vom Nutzer über ein Kontextmenü bearbeiten zu lassen.
Verschiedenste *Markup Interfaces* sind bereits vorhanden, welche definieren, wann es sich bei einem Element um eine *Connection*, einen *Connection Point* oder um eine *Input*- (Senke) oder *Output Component* (Quelle) handelt. Beide Components sind über das *Component*-Interface miteinander vereint, und Components, Connections und Connection Points implementieren als darstellbare Elemente zudem das Markup Interface *Presentable*. Als Elemente, welche konfigurierbare Eigenschaften enthalten können, betten Components und Connection Points außerdem das Interface *Configurable* ein.
Aktionen wie das Löschen oder Hinzufügen von Komponenten prozessieren intern über ein eigens kreiertes System von so genannten *Commands*, welche an den *FlowController* gereicht werden, um dort vom Typen des Commands abhängige Logik auszuführen und Aufgaben zu delegieren. Der FlowController stellt dementsprechend die zentrale Service-Klasse dar, welche für den Großteil der internen Business-Logik zuständig ist.

4.3 Architektur

Vieles der vorhandenen Architektur kann direkt oder mit Abänderungen weiterverwendet werden.
Dennoch ist das System weitreichend zu erweitern, um eine umfangreiche Arbeitsplatz-Verwaltung, eine generische Komponenten-Architektur mit variablen Eigenschaften der Komponenten oder eine Historisierung von Aktionen bieten zu können und zudem starken Wert auf Usability der Anwendung für den Nutzer legen zu können.

Als Basis muss zunächst die Gesamtarchitektur überarbeitet werden. Nebst der Erweiterung um **Maven** erfordert die gewünschte Wartbarkeit und Erweiterbarkeit des Systems eine zugrundeliegende **MVC**[21]**-Architektur**. Die diversen zu implementierenden Ansichtselemente wie der Editor, die MenuBar oder diverse Modal[22]-Elemente fungieren dementsprechend als *View*.
Ein zu implementierendes *ClientModel* wird sämtliche Daten enthalten, welche für den ordnungsgemäßen Ablauf des Systems notwendig sind. Unter anderen werden solche Daten etwa der aktuell verwendete Arbeitsplatz, notwendige, auf Client-Seite zu haltende Informationen über den Nutzer oder das dem Nutzer verfügbare Repertoire an Komponenten sein.
Zudem ist ein *ClientController* zu erstellen, welcher die zentrale Schnittstelle aller Bereiche der Anwendung für die Geschäftslogik ist. Hier wird auf sämtliche Aktionen aus der View gelauscht, welche dann weiterverarbeitet und an andere Komponenten wie den Editor oder auch den FlowController delegiert werden.
ClientModel und ClientController werden als Singleton existieren, da die Verfügbarkeit beider Instanzen in allen Bereichen der Anwendung gegeben sein muss und zudem auch nur als Unikat bestehen dürfen.

Aufrufe an den Server, etwa zur Persistierung von Arbeitsplätzen oder zu Login-Versuchen, geschehen über zentrale **Service**-Schnittstellen. Diese sind Subinterfaces der *RemoteService*-Schnittstellen und ermöglichen eine asynchrone Kommunikation mit dem Server über das *Remote Procedure Call*-Protokoll, auf welchen etwa auch das Java RMI-Protokoll aufbaut. Dieses ist gemäß der festen Implementierung in GWT eine sehr komfortable Möglichkeit, um Aufrufe schnell (weiter-)entwickeln zu können. Dank der Asynchronität erfährt der Nutzer zudem weitestgehend nicht, wann Aufrufe an den Server ablaufen und wird daher auch nicht von solchen in der Bedienung gestört.
RemoteServices erfordern das zusätzliche Erstellen von so genannten *Async*-Interfaces, welche jeden in der Service-Klasse erstellten Methodenkopf ebenso abbilden, mit dem Unterschied, dass diese einen zusätzlichen Parameter

[21] Model View Controller
[22] vergleichbar mit Popups

in Form eines *AsyncCallbacks* erfordern. Diese AsyncCallbacks enthalten die Funktionen *onSuccess(..)* und *onFailure(..)*, mit welchen gegebenenfalls Rückgabewerte gehändelt werden können, weiteres Vorgehen definiert oder Fehlermanagement betrieben werden kann.
Auf Server-Seite existiert dann die Implementierung der Service-Schnittstelle. Jede zu erstellende Implementierungs-Klasse hat die *RemoteServiceServlet*-Klasse zu erweitern und wird in der *web.xml*-Datei eingetragen und somit für Aufrufe verfügbar gemacht.

Bisher bestehende Commands, welche dem FlowController Aufgaben zur Erstellung oder Entfernung von IVK-Komponenten übermitteln, werden künftig durch GWT-Events ersetzt. Dies erscheint, für eine einheitliche Architektur, als notwendig, da sämtliche durch das UI stattfindenden Aktionen ebenso in Form von Events an den ClientController übermittelt werden. Über einen im ClientModel deponierten *EventBus* kann dieser auf solche Events wie etwa Klick-Events oder Arbeitsplatz-Verarbeitungs-Events lauschen und dementsprechend delegieren.

Die **Generalisierung von Komponenten**, um flexibles Hinzufügen neuer Komponenten mit beliebiger Logik und deren Veränderbarkeit zu ermöglichen, erfordert zudem eine Überarbeitung des bisher vorhandenen Systems, welches lediglich fest implementierte, vordefinierte Komponenten zuließ.
Systembezogene Logik muss ausgelagert werden und statt vorgefertigter Komponenten müssen so genannte Templates von Komponenten erstellt werden, welche die abstrakten Komponenten-Klassen implementieren. Diese existieren in drei Varianten als *InputComponentTemplate*, *OutputComponentTemplate* und dem neu zu erstellenden *InputOutputComponentTemplate*, welches die Transformationen mit Informationsein- und -ausgang darstellt. Diesen wird bei Instanziierung, also bei Hinzufügen in den Editor, lediglich eine Konfiguration in JSON-Form übergeben, welche sämtliche Funktionalität der Komponente definiert.
Jene Komponenten-Templates werden erstellt von einer *ComponentFactory*, welche mittels statischer Funktionen aus einem übergebenen JSON-Objekt ein in der Komponentenvariante spezifisches Component-Objekt aus den Templates erzeugt und zurückgibt. So besteht für aufrufende Klassen keine Notwendigkeit, etwas über die Beschaffenheit der Templates zu wissen, wodurch sie sich in der Erzeugung voll auf die Fabrik-Klasse verlassen können.

Für eine verfügbare **Historisierung** der Anwendung, also der Funktion des Widerrufens und Wiederholens von Befehlen, ist zudem ein *ExecutionHistoryController* zu erstellen, dessen Instanz im ClientModel gehalten wird und vom FlowController über sämtliche Aktionen unterrichtet wird. Er sammelt dann diese Aktionen in eigenen Listen und kann bei Aufruf die zuletzt durchgeführte Aktion widerrufen, indem er ein jeweils konträres Event feuert. Die-

se widerrufene Aktion wird anschließend in einer eigenen Menge gespeichert, wo sie dann im Falle eines Wiederholen-Aufrufs wieder konträr ausgeführt werden kann.

Es muss ebenso eine **Workspace**-Komponente entwickelt werden, welche die zentrale Verwaltung des Arbeitsplatzes und die Sammlung und Preisgabe der im Arbeitsplatz enthaltenen Elemente übernimmt. Sie ist zudem anhand von intern verwalteten *HashMaps* in der Lage, Zwischenstände vom zuletzt ausgeführten Persistieren zum aktuellen Stand des Arbeitsplatzes zu halten, um bei Bedarf daraus ein gesamtes Delta-Objekt zu erzeugen, welches an den Server für eine dynamische Speicherfunktion mit minimaler Datenmenge gesendet werden kann.
Die Delegation zum Erzeugen eines Deltas wird architekturbedingt im ClientController geschehen.

Im Bereich der Views werden bereits vorhandene Widgets insofern erweitert, dass sie ihr Layout mittels **UiBinder** aufbauen. Dank des deklarativen Ansatzes in Form von XML kann hier, entgegen des programmatischen Stils, eine deutlich bessere Wart- und Erweiterbarkeit der Widgets erzielt werden. Neu erstellte Widgets werden, sofern aufgrund des Umfangs der im Widget enthaltenen Elemente und der Deklarierbarkeit dieser Elemente sinnvoll, dementsprechend ebenso mit Verwendung des UiBinders implementiert.
Die Editor-Komponente wird nicht umfangreich überarbeitet, sondern nur um grundlegende Features erweitert. So wird hier ein **Key-Handling** eingeführt, welches die Aufrufe von Funktionen über getätigte Tastenkombinationen erwirkt. Desweiteren wird die Funktion zur Darstellung des Editor-Hintergrundes gekapselt, um sie regelmäßig in Abhängigkeit von Zoom- und Editor-Positions-Parametern aufrufen zu können und den Hintergrund neu zu zeichnen.
Für die erforderte flexible Mehrsprachigkeit des Systems wird ein **i18n**[23]-System eingeführt. Dieses erweitert die GWT-eigene *Messages*-Komponente, welche die für unsere Ansprüche notwendige Möglichkeit bietet, parametrisierte Ausgaben zu erzeugen. Zudem bieten Messages die Möglichkeit, einfache *properties*-Dateien nutzen zu können, welche die für ihre Sprache vorgefertigten Strings enthalten und dementsprechend besonders leicht zu warten sind. Mittels des *Deferred Bindings* werden zur Kompilierzeit nutzbare Klassen alle verfügbaren Sprachen erzeugt und zur Ladezeit genau nur die eine Sprache geladen, welche systemweit genutzt werden soll.
Abschließend wird das System um eine **Logging**-Architektur erweitert. Es soll eine Adapter-Klasse entstehen, welche die für GWT zugängliche Logger-Klasse aus dem java.util-Paket umschließt und eine komfortable, aus *log4j*[24]

[23] Internationalization
[24] http://logging.apache.org/log4j/2.x/

bekannte Logging-Klasse darstellt, welche einfach erzeugt werden kann und anhand simpler Funktionen wie *warn(..)*, *info(..)* oder *debug(..)* einheitliche Log-Ausgaben hervorbringt.

Eine UML-Ansicht über die systemrelevantesten Komponenten mit einer gekürzten Menge an dargestellten Funktionen je Klasse ist im Folgenden ersichtlich:

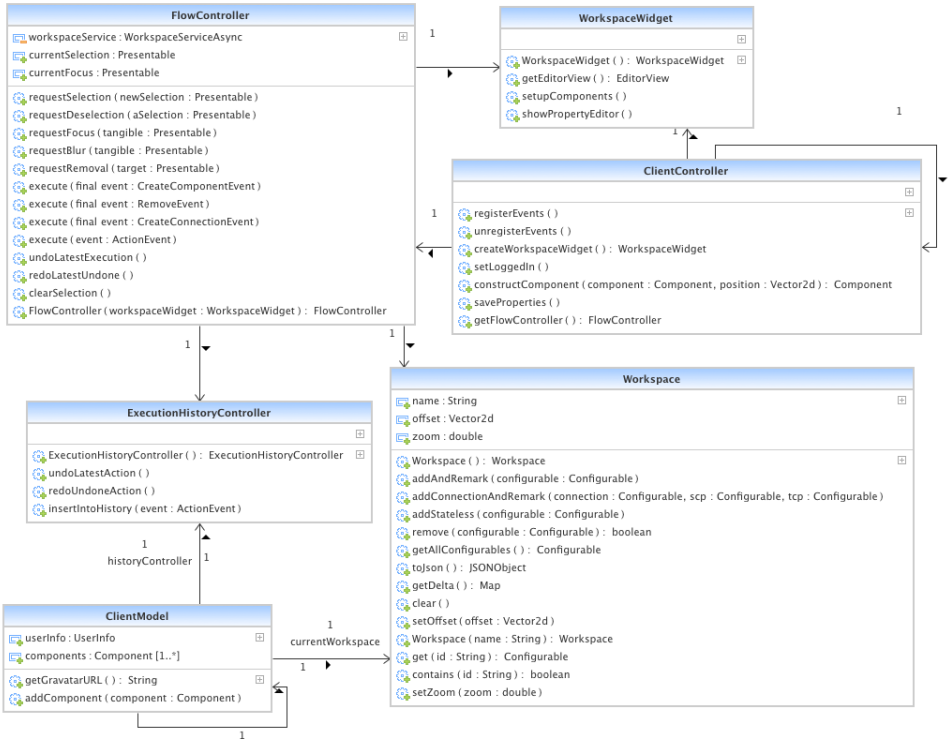


Abbildung 19: Mangold-Architektur im UML-Diagramm

5 Umsetzung

5.1 Verwendete Technologien

Für die Umsetzung der genannten Anforderungen wurden nebst dem Google Web Toolkit folgende weitere diverse Technologien und Frameworks benötigt und genutzt.

5.1.1 Grundsätzliche Werkzeuge

Unterstützend für die Entwicklung wurde für die Versionierung das SCM[25]-Tool **Git** verwendet, speziell der Git-Repository Host *Bitbucket* von Atlassian[26].
Zusätzlich wurde für die Erstellung und den Bau des Prototypen auf das Build-Management-Tool **Maven**[27] und dessen *Maven GWT Plugin*[28] in der Archetype-Version 2.6.0 zur Generierung eines neuen GWT-Projekts mit Maven zurückgegriffen.
Mit der Unterstützung von Maven wird auch das Continuous Integration-Tool **Jenkins**[29] und das Code Quality-System **Sonarqube**[30] (ehemals *Sonar*) verwendet.
So wird bei einem getätigten *Push* auf das Bitbucket-Repository ein sogenannter Hook auf den Jenkins-Server ausgelöst, welcher sich anschließend die Updates von Bitbucket besorgt und mittels Maven das Projekt kompiliert. Wenn der Build erfolgreich verlief, wird das System via Sonarqube auf Qualität untersucht und zusätzlich auf dem lokalen Application Server **Tomcat**[31] deployt.

5.1.2 Weitere Frameworks und Bibliotheken

Die Basistechnologie, auf welcher der Editor aufbaut, ist das **Canvas-Element**, welches mit HTML5 standardisiert wurde. Canvas, zu Deutsch *Leinwand*, ist ein HTML-Element, mit welchem via diverser Scriptsprachen, meist JavaScript, dynamisch Bitmap-Grafiken gerendert und in HTML-Dokumente integriert werden können. Mit Canvas können neue Grafiken erstellt, Bilddaten manipuliert oder auch Animationen erstellt werden[30, 40].
Auf Canvas aufbauend wird für diesen Prototypen das freie JavaScript-Framework **kinetic.js** - respektive dessen GWT-Portierung **kineticGWT**[32] - verwendet. Dieses Framework vereinfacht die Erstellung von Elementen

[25] Source Code Management
[26] Repository-URL: *https://bitbucket.org/sreim/mangold*
[27] siehe *http://maven.apache.org/* für nähere Erläuterungen
[28] *http://mojo.codehaus.org/gwt-maven-plugin/user-guide/archetype.html*
[29] *http://jenkins-ci.org/*
[30] *http://www.sonarqube.org/*
[31] http://tomcat.apache.org/
[32] *https://github.com/neothemachine/KineticGWT*

und Animationen und die Umsetzung von Event-Handlings innerhalb eines Canvas-Elements. So können mit wenig Aufwand Flächen und Körper gezeichnet werden, welche verschiebbar innerhalb des Canvas sind und diverse Klick- und Drag-Events feuern, auf die reagiert werden kann.

Bootstrap ist ein von einem Twitter-Entwickler im Jahr 2010 veröffentlichtes JavaScript-Front-End-Framework und Open Source-Projekt, welches umfangreiche Vorlagen an Styles und Komponenten für grafisch ansprechende Oberflächen bietet und sich mittlerweile zu einer der bedeutendsten Vorlagen für modernes Webdesign entwickelt hat [1]. Für das Google Web Toolkit existiert zudem eine eigene Portierung namens **GWT-Bootstrap**[33]. Bootstrap wird für diesen Prototypen als designerische Grundlage verwendet und nebst des grundlegenden Templates auch für die Abbildung diverser Komponenten wie der Navigationsleiste, den modalen Popups oder den Tab-Bars eingesetzt.

Zudem werden für diverse vereinzelte Komponenten und Funktionen weitere entwicklungsvereinfachende Tools genutzt.
Das umfangreiche Framework **GWTQuery**[34], speziell das Plugin *gquery-dnd-bundle*, wird im Prototypen für die Drag&Drop-Funktionalität aus der Navigation in das Canvas-Element eingesetzt. GWTQuery ist ein Projekt mit dem Ziel, die bekannte JavaScript-Bibliothek *jQuery*[35] in GWT zu implementieren.

Desweiteren wird, für die Abbildung des Komponenten-Prozess-Editors, auf die JavaScript-Bibliothek **Codemirror**[36], speziell auf den GWT-Wrapper **Codemirror2-GWT**[37], zurückgegriffen. Diese bietet einen umfangreichen Editor für Programmcodes in diversen Programmiersprachen, einschließlich der Funktionen des Syntax-Highlightings, Auto Completion oder Code Folding.

5.2 Implementierung

Nachdem im Kapitel 4.1 die notwendigen Anforderungen besprochen wurden, ist es nun die Aufgabe, die tatsächliche Umsetzung des Prototypen zu realisieren. Zugunsten der Übersichtlichkeit wird sich im Anschluss auf die jeweiligen einzelnen Teilbereiche des Prototypen bezogen, darunter den Editor (Kapitel 5.2.1), der Umsetzung der Komponenten (Kapitel 5.2.2), sowie der Admin-Oberfläche (Kapitel 5.2.3).

[33] http://gwtbootstrap.github.io/
[34] *https://code.google.com/p/gwtquery-plugins/*
[35] *http://jquery.com/*
[36] *http://codemirror.net/*
[37] *https://code.google.com/p/codemirror2-gwt/*

5.2.1 Editor

Das Hauptelement des Prototypen ist der Editor. Dies ist die Funktionalität, mit der Komponenten bearbeitet werden, Arbeitsumgebungen geladen und gespeichert werden und innerhalb der Arbeitsfläche navigiert werden kann. Ein erster Einblick in die entworfene Oberfläche kann in Abbildung 20 gewonnen werden.

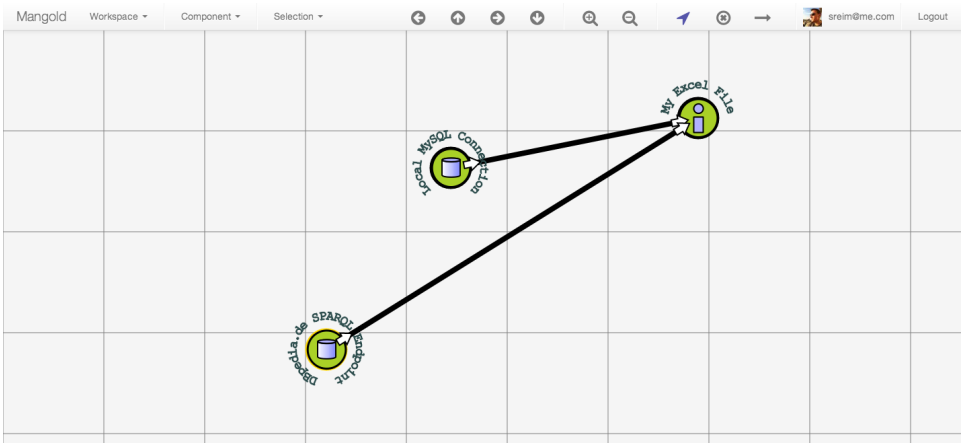


Abbildung 20: Mangold - Editor Übersicht

Wie dort zu sehen, enthält der Editor oberhalb des Bildes eine Navigationsleiste. Im ausklappbaren Menü *Workspace* erscheinen die Möglichkeiten des Ladens und Speicherns eines Arbeitsplatzes, sowie die Funktionen *Undo* und *Redo*, also das Widerrufen und Wiederholen eines Vorgangs.
Innerhalb des Navigationspunktes *Component* erscheinen sämtliche verfügbaren und wählbaren Komponenten, welche zuvor in der Verwaltungsoberfläche (siehe Verwaltungsoberfläche, Kapitel 5.2.3) erstellt worden sind. Die ausgeklappte Ansicht der Component-Navigation ist in Abbildung 21 zu sehen.

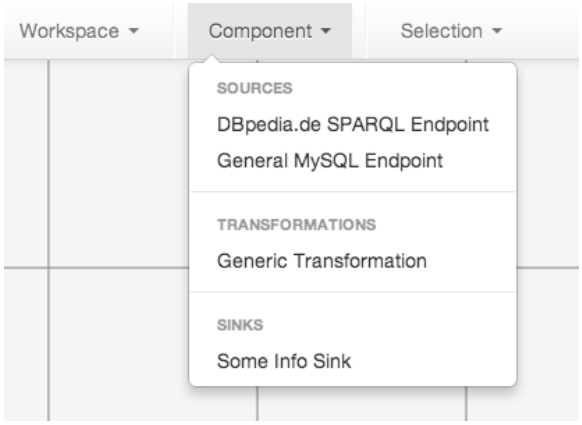


Abbildung 21: Mangold - Navigation Components

Zudem erscheint bei Auswahl einer IVK-Komponente ein drittes Navigationselement, das *Selection*, mittels welchem die zugehörigen Parameter der Komponente modifiziert werden können.
Außerdem enthält die Navigationsleiste Funktionen für die Navigation innerhalb des Arbeitsbereiches: Jeweils Buttons für die Navigation nach oben, unten, links und rechts, sowie Buttons für das Vergrößern, bzw. Verkleinern des sichtbaren Bereichs der Arbeitsfläche. Zusätzlich finden sich drei weitere Buttons zum Anwählen und Verschieben, zum Löschen und zum Verknüpfen von Komponenten. Sofern der Nutzer sich hinzukommend mit einer E-Mail-Adresse im System angemeldet hat, welche er zuvor auf der Plattform *Gravatar*[38] hinterlegt hat, wird in der Navigationsleiste desweiteren sein gespeichertes Profilbild dargestellt.

Komponenten können dem Editor hinzugefügt werden, indem der Nutzer entweder direkt auf eine Komponente klickt oder eine Komponente aus der Liste in den Editor zieht. Jedes Komponenten-Navigationselement besitzt daher nicht nur einen *ClickListener*, welcher dafür sorgt, dass an einem fixen Punkt im Editor diese Komponente neu erzeugt wird, sondern wird mithilfe des *GWTQuery*-Frameworks vom Wrapper *DraggableWidget* umgeben und kann dann in das nicht sichtbare *DroppableWidget* platziert werden, welche die eigentliche Editor-Komponente beherbergt.

Die Navigation innerhalb des Editors, also das Springen in alle Himmelsrichtungen und das Heran- und Wegzoomen im Editor, wird nach dem Klick auf einen Navigations-Button durch eine Neuzeichnung des Editors und Repositionierung der Komponenten und Verknüpfungen realisiert. Dies geschieht

[38] Ein externer Service zum Hinterlegen von Profilbildern für vom Nutzer angegebenen E-Mail-Adressen - http://de.gravatar.com/

in Abhängigkeit von Zoom-Level und den x- und y-Abständen vom Koordinatenursprung.

5.2.2 Komponenten

Für eine möglichst generische Implementierung von Komponenten einer IVK, also den Quellen, Senken, Transformationen und deren Verbindungen miteinander, ist es notwendig, weitestgehend von Programmiersprachen und deren Limitierungen unabhängig zu sein. Es wäre nur schwierig umsetzbar, eine vielfach funktionale Präsenz von Komponenten anzubieten, welche sämtliche erdenklichen Eventualitäten unterstützt, wenn die Implementierung dieser Komponenten an sprachliche Gegebenheiten, wie vorgegebene Schnittstellen und Funktionsdefinitionen, gebunden ist. Aus diesem Grund wird dem Nutzer selbst die Möglichkeit gegeben, die Implementierung der Komponente vorzuschreiben, indem er der Komponente eigenen in *Groovy* oder *Java* entwickelten Code zugänglich macht.

Dies ist dadurch möglich, dass eine Komponente intern nichts anderes ist als ein JSON-Objekt. Die Komponente enthält dadurch neben dem eigentlichen Prozess-Code, welcher vom Nutzer definiert wird, lediglich Informationen zu ihrem Namen, den vom Code benötigten Parametern und Informationen über deren Eingabe, bzw. Ausgabe. Quellen haben generell ein Feld *Output*, Senken haben stets ein Feld *Input*. Ein Beispiel für eine Quelle, welche eine MySQL-Datenbank abfragen soll, wird in Abbildung 22 gezeigt. Aus Gründen der Übersichtlichkeit wird an dieser Stelle auf den Prozesscode verzichtet.

```
{
  "name" : "General MySQL Endpoint",
  "host": {
    "default": "http://127.0.0.1:5050",
    "type": "textbox"
    },
    "database": {
      "default": "mainDb",
      "type": "textbox"
    },
    "user": {
      "default": "root",
      "type": "textbox"
    } ,
    "password": {
      "default": "password",
      "type": "textbox"
    },
    "output": {
      "name" : "MySQL Source Connection Point",
      "query" : {
        "default" : "SELECT foo FROM bar",
        "type" : "textbox"
      },
      "process" : "[...]"
    }
}
```

Abbildung 22: Beispiel-Komponente in JSON

Dies ist somit schon alles, was eine generische Komponente benötigt.
Die Felder *name* und *input*, bzw. *output* müssen in jeder Komponente gesetzt sein. Felder wie *host* oder *database* sind optionale Parameter, wobei jeder Parameter ein Feld *default* benötigt, welches entweder einen beliebigen Standard-Text setzt oder ggf. die vom Anwender zu tätigende Eingabe näher spezifiziert.

Zusammen mit den JSON-Snippets aller anderen verfügbaren Komponenten wird diese nach dem Login des Clients proaktiv asynchron vom Client abgeholt. Jedes JSON-Objekt wird sicherheitsbedingt vorher vom Server in eine serialisierbare Wrapper-Instanz gelegt. Sobald der Client die Komponenten-Menge erhalten hat, formt dieser jedes JSON-Objekt zu einer instanzierbaren und für ihn nutzbare Klasse um. Dies geschieht im Kern in der sogenannten *ComponentFactory*, welche anhand der im JSON gegebenenfalls enthaltenen Felder *input* und *output* intern entscheidet, ob es sich um eine Quelle, eine Senke oder eine Transformation (sowohl *input* als auch *output*) handelt. Abhängig von dieser Entscheidung wird eine von drei Template-Klassen genutzt und instanziert, indem sie das gesamte JSON-Objekt der Komponente aufnimmt. Auf diese Art lässt sich bei Gebrauch solch einer Komponente eine weitere neue Instanz der Komponente erzeugen, mit der flexibel im System gearbeitet werden kann. Codeintern lautet die Bezeichnung für Senken *Input Component*, Quellen werden *Output Component* genannt und Transformationen werden als *Input Output Component* bezeichnet. Im Abbildung 23, wird das InputOutputComponentTemplate in gekürzter Form dargestellt, welches also ein Gerüst einer Transformation symbolisiert.

```java
public class InputOutputComponentTemplate extends
    AbstractInputOutputComponent {
  private JSONObject json;
  private JSONObject outputJson;
  private JSONObject inputJson;

  public InputOutputComponentTemplate() {
    super();
    this.json = new JSONObject();
    this.json.put("name", new JSONString(I18N.Util.get
        ().newComponent()));
    this.json.put("input", new JSONObject());
    this.json.put("output", new JSONObject());
  }

  public InputOutputComponentTemplate(JSONObject json)
      {
    this();
    this.json = json;
  }

  public Image getSymbolImage() {
    JSONString strImage = null;
    Image img = null;

    if(json.get("symbolImage") != null && (strImage =
        json.get("symbolImage").isString()) != null) {
      img = new Image(strImage.stringValue());
    }
    if(img == null)
      img = new Image(TheResources.INSTANCE.
          iconDataSource().getSafeUri());
    return img;
  }

  protected void setupConnectionPointConfig() {
    JSONValue inputVal = this.configuration.get("input
        ");
    JSONValue outputVal = this.configuration.get("
        output");
    if(inputVal != null) {
      inputJson = inputVal.isObject();
    }
    if(outputVal != null) {
      outputJson = outputVal.isObject();
    }
  }
}
```

Abbildung 23: Die InputOutputComponent

Nachdem alle Komponenten umgeformt wurden, werden diese in das allerorts zugängliche *ClientModel* (Vgl. Kapitel 5.2.4) gelegt.
Sobald der Nutzer eine der Komponenten dem Editor hinzufügt, wird in der *ComponentFactory* eine neue Grundinstanz des jeweiligen Templates aufgebaut. Dieser werden anschließend die JSON-Daten der Komponente (intern als *FieldConfiguration* bezeichnet) zugesteckt und schließlich in den Editor gezeichnet und mit Event-Listeners und Animationsinformationen belegt.
Für weitere Informationen, wie dieser JSON-Code erzeugt wird und wie die Standard-Werte und Prozess-Informationen gestaltet werden, ist der folgende Abschnitt Verwaltungsoberfläche, Kapitel 5.2.3 zurate zu ziehen.

5.2.3 Verwaltungsoberfläche

Die Verwaltungsoberfläche ist in Mangold zu erreichen, indem der Nutzer als Nutzername und Passwort den Begriff *admin* eingibt. Die Darstellung der sich anschließend öffnenden Oberfläche ist in Abbildung 24 ersichtlich.

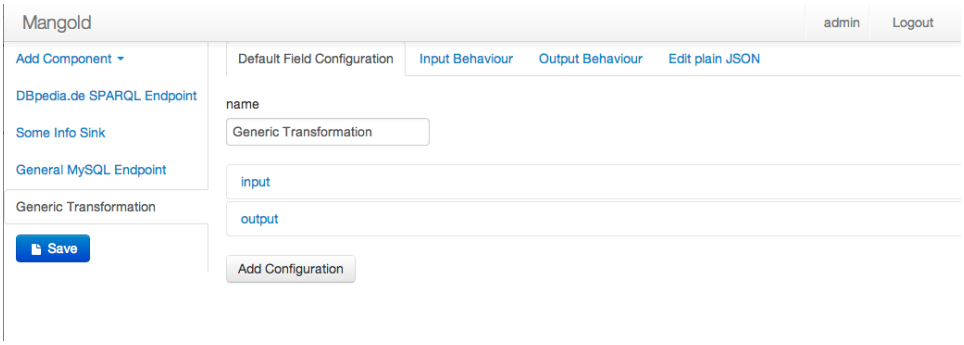


Abbildung 24: Die Verwaltungsoberfläche

Innerhalb dieser Oberfläche werden auf der linken Seite sämtliche verfügbaren Komponenten aufgelistet, welche zuvor bei erfolgreichem Login vom Server geholt und in das *ClientModel* (Vgl. Kapitel 5.2.4) geladen wurden. Jede Komponente enthält drei, in Tabs aufgelistete, Einstellungskategorien: Die *Default Field Configuration*, das *Component Behaviour* und das Feld *Edit Plain JSON*. Zusätzlich kann durch Klick auf das Navigationselement *Add Component* eine Komponente hinzugefügt und über den Button *Save* der gesamte Satz an aufgelisteten Komponenten zurück an den Server geschickt, und dort gespeichert, werden.

Im Reiter **Default Field Configuration** lassen sich generelle Einstellungen treffen bezüglich des Namens der Komponente, den von ihr benötigten Parametern (als *Configuration* bezeichnet) inklusive deren Default-Werten und dem jeweiligen Verbindungspunkt mitsamt eigener Parameter. Hier können unbegrenzt *Configurations* hinzugefügt, sowie bestehende gelöscht werden. Für den Aufbau dieses Panels wird die JSON-Konfiguration der Komponente herangezogen. Zusätzlich wird im Hintergrund ein Baum aufgebaut, welcher für jeden Konfigurationsteil einen neuen Knoten mit dem Feldnamen und dem enthaltenen Wert des Teiles enthält. Abbildung 25 verdeutlicht diese Baumstruktur anhand des Beispiels einer MySQL-Quellkomponente. Aus diesem Baum kann beim Speicherprozess rekursiv der neue, vom Nutzer gegebenenfalls geänderte Wert, ausgelesen und wieder zurück in validen JSON-Code umgewandelt werden.

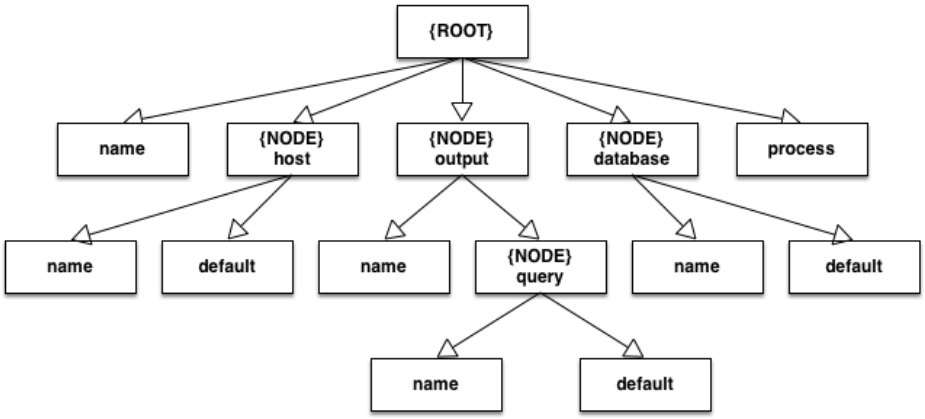


Abbildung 25: Beispielhafte Baumdarstellung einer MySQL-Komponente

Der Reiter **Component Behaviour** erlaubt das Erstellen des, bzw. der Prozesscodes der Komponente. Da der Prozess davon abhängt, ob es sich um eine Quell- oder eine Senken-Funktion handelt, besitzt eine Transformation sogar zwei voneinander unabhängige Prozesse. Der hier eingegebene

Code wird nach dem Speichern oder Wechseln des Tabs in das Feld *process* innerhalb des jeweiligen Knotenpunktes (also *input* oder *output*) der Komponente geschrieben. Dieser Prozesscode ist impliziter Teil einer übergeordneten Methode. Auf dem Server wird dieser Code in diese Methode eingebunden und dann ausgeführt. Daher ist das Ergebnis des Prozesscodes einer Komponente stets über das *return*-Statement zurückzugeben. Ein beispielhafter Prozesscode ist in Abbildung 26 zu sehen. Wie zu erkennen, ist dieser Prozess in der Sprache *Groovy*[39] geschrieben. Von der ausführenden Plattform, welche dem Editor künftig zur Verfügung gestellt wird, wird erwartet, dass sie in der Lage ist, die wichtigsten Sprachen der Java Virtual Machine, also Java, Groovy und Scala, zu verstehen.

```
def sql = Sql.newInstance("jdbc:mysql://${host}/${
   database}", "${user}", "${password}", "com.mysql.
   jdbc.Driver");

def result = [];
sql.eachRow(${node.query}) {
    result << it.toRowResult();
}
return result;
```

Abbildung 26: Beispielhafter Prozesscode einer Pseudo-MySQL-Komponente

Innerhalb des Prozesscodes kann auf die Eingaben für die Parameter einer Komponente zugegriffen werden, indem der Name des Parameters mit einem Dollarzeichen und geschweiften Klammern umrahmt wird. An die Parameter eines zugehörigen Verbindungspunktes gelangt man über den Bezeichner *node*.
Im Reiter **Edit Plain JSON** lässt sich der gesamte JSON-Konfigurations-Code direkt editieren. Nach Abschluss der Bearbeitung wird der eingegebene JSON-Code validiert.

5.2.4 User-Management und Workspaces

Das System *Mangold* wurde mit Rücksicht auf Nutzerorientierung und Arbeitsplatzbindung gestaltet.
Unter **Nutzerorientierung** ist die Notwendigkeit der Eingabe von Nutzerdaten zum Zugriff auf das System zu verstehen. Dazu existiert eine einfache Login-Seite, welche nach Eingabe von Nutzername und Passwort die Daten an den Server schickt. Dieser persistiert den Nutzernamen in der Server Session und schickt dem Nutzer ein einfaches User-Objekt zurück. Der Client

[39] Eine die Funktionalitäten von Java und Ruby vereinende Sprache für die Java Virtual Machine

vermerkt sich dieses Objekt anschließend im *ClientModel* und baut sich die Oberfläche auf. Bei einem gegebenenfalls stattfindenden Wiedereintritt auf das System nach kurzzeitiger Abwesenheit schaut der Server in seiner Session nach, ob der User bereits existiert und gibt ihm im Falle dessen ein neues User-Objekt mit seinem Nutzernamen zurück, ohne dass er sich erneut manuell einloggen muss.

Das **ClientModel** ist ein Singleton, welches die regelmäßig und in verschiedenen Komponenten benötigten Nutzdaten enthält. So beherbergt dieses sowohl das User-Objekt, welches den Nutzernamen und Informationen zum Nutzertyp enthält, als auch die Menge nutzbarer Komponenten und den aktuell genutzten Arbeitsplatz.

Ein Arbeitsplatz (**Workspace**) steht hier für die Menge aller Komponenten und Elemente einer Informationsverarbeitungskette und deren zugehörigen Einstellungen und Anpassungen. Ein Arbeitsplatz ist also all das, was innerhalb des Editors positioniert und bearbeitet wird.
Nach dem Login wird intern ein sogenannter Default-Workspace erstellt, welchem die im Editor eingestellten Komponenten und Verknüpfungen hinzugefügt werden. Über den Menüpunkt *Save* kann dieser Arbeitsplatz dann auf dem Server gespeichert und jederzeit via *Load* geladen werden. Dazu wird dauerhaft ein *Workspace*-Objekt im *ClientModel* fixiert und bei jedem Hinzufügen, Löschen und Ändern von Komponenten und Verbindungen im Editor konsistent zum Editor-Inhalt gehalten. Beim Speichern eines Arbeitsplatzes werden sämtliche relevanten Informationen aller Komponenten, also sowohl deren Feld-Konfiguration (Prozesscode, Name, Parameter), als auch generelle Daten wie Position der Komponente im Editor oder die Verbindungen zu anderen Komponenten, zusammengefasst und an den Server gesendet, welcher dann die Informationen auf Basis des Workspace-Namens persistiert.
Wenn der Nutzer einen existenten Arbeitsplatz laden möchte, sendet der Server nach der Anfrage die Liste der Konfigurationen zurück an den Browser, welcher im Anschluss aus der JSON-Liste wieder vollwertige Komponenten formt (Vgl. Kapitel 5.2.2 für nähere Informationen bzgl. des Bauens von Komponenten-Klassen aus JSON-Objekten).
Ebenso ist Mangold aber auch in der Lage, sogenannte Delta-Daten dauerhaft im Hintergrund zu persistieren, ohne dass es einer proaktiven Aktion des Nutzers bedarf. Dazu werden die getätigten Aktionen in einer von drei Maps im Workspace geordnet - den hinzugefügten, den gelöschten und den editierten Elementen. Wenn also etwa der Name einer bereits existenten Komponente im Editor geändert wird, dann wird diese Komponente in die Map der Änderungen gepackt. Wird anschließend eine neue Komponente in den Editor gezogen und mit der bestehenden Komponente verknüpft, dann wird diese und die Verbindung beider in die Map der hinzugefügten Elemente

gelegt. Der Arbeitsplatz wird dann regelmäßig nach existenten Delta-Daten abgefragt. Sofern solche existieren, sendet der Client diese drei Maps an den Server und setzt seine Delta-Daten wieder zurück. Anhand der Information, welche Elemente hinzugefügt, geändert oder gelöscht wurden, kann der Server dann entsprechende Datenbankmanipulationen durchführen. Hinzugefügte Elemente werden der Datenbank hinzugefügt genauso wie gelöschte Elemente auch von der Datenbank gelöscht werden; Editierte Komponenten erfahren auf der Datenbank ein Update-Statement.

5.2.5 Internationalization

Der Prototyp unterstützt ebenso die **Internationalisierung** (i18n), also die sich dynamisch anpassende Sprachausgabe. Aktuell unterstützt Mangold die beiden Sprachen Englisch und Deutsch, während bei fehlender Festlegung Englisch als Standard-Sprache ausgewählt wird. Die Sprache Deutsch kann ausgewählt werden, indem der URL der Parameter *locale=de* hinzugefügt wird.

Technisch erreicht wird dies grundsätzlich mithilfe zweier Property-Dateien - *I18N_de.properties* und *I18N_en.properties* - und eines Interfaces namens I18N, welches die Keys definiert, über welche auf die jeweilige Übersetzung zugegriffen werden kann. Dieses Interface erweitert das GWT-*Messages*-Interface, welches für eine kompilierzeitliche Einbindung von Nachrichten aus verschiedenen Quellen (die Property-Dateien) sorgen kann.
Dieses Interface wird in GWT mithilfe des *Deferred Bindings* erzeugt, wodurch sämtliche Sprachversionen des I18N-Interfaces kompiliert werden (also Englisch und Deutsch), aber nur eine, nämlich die eine benötigte Sprache, vom Client geladen wird.
Ein kleiner Bereich des simplen I18N-Interfaces ist in Abbildung 27 nachzuvollziehen.

```
public interface I18N extends Messages {
  String about();
  String load();
  String save();
  String addComponent();
  String addFieldConfiguration();
  String remove();
  String configure();
  String undo();
  String redo();
  String successfullySaved();
  String numberComponentsReturned(int noComponents);
}
```

Abbildung 27: Das I18N-Interafce

6 Evaluierung

Im Anschluss an die abgeschlossene Umsetzung des Prototypen soll nun verifiziert werden, inwieweit die Anforderungen an den Editor umgesetzt wurden und welche Mängel er gegebenenfalls noch aufweist. Zudem prüfen wir die Anwendung im Hinblick auf die Generizität der Komponenten und erörtern die Nutzbarkeit der Anwendung.

6.1 Validierung der Anforderungen

In dieser Arbeit entstand eine Anwendung, welche in der Lage ist, beliebige Mengen an Komponenten einer Informationsverarbeitungskette zu einem Editorbereich hinzuzufügen und diese miteinander zu verknüpfen. Wie angefordert können diese Komponenten innerhalb des Editors mittels Drag&Drop verschoben werden und Verbindungen zweier Komponenten erstellt und wieder entfernt werden. Vordefinierte Parameter von Komponenten können über ein separates Menü ausgefüllt und modifiziert werden.
Der Editor ist zudem beweglich. Das bedeutet, der Nutzer ist hier in der Lage, sich innerhalb des großen Editorbereichs in allen Himmelsrichtungen entlang zu bewegen und den Sichtbereich bei Belieben zu vergrößern und zu verkleinern.

Ebenso ist das Projekt *Mangold* in der Lage, mit einem Server zu kommunizieren. Es wurde ein User-System implementiert, welches dem Nutzer die Notwendigkeit einer Anmeldung zu Beginn der Benutzung auferlegt. Abhängig von den Login-Daten gibt der Server die Information zurück, ob der Nutzer ein gewöhnlicher User oder ein Administrator ist. Dennoch ist der Server gemäß der Anforderungen noch nicht komplex genug. Er ist bisher noch nicht in der Lage, einem Nutzer die Nutzung der Plattform zu verwehren, da er zum aktuellen Zeitpunkt noch nicht über ein Registrierungssystem verfügt. Dies bedeutet, dass Nutzerdaten bisher nicht prüfbar in einer gesonderten Datenbank existieren, sondern per sé lediglich in die Server-Session geschrieben werden. Daher bietet auch die Oberfläche keine Möglichkeit, sich als neuer Nutzer zu registrieren. Der Prototyp lässt beliebige Login-Daten zu und informiert ausschließlich über gesonderte Administrator-Funktionalitäten.

Bezüglich der angeforderten Arbeitsplatzverwaltung bietet der Prototyp umfangreiche Funktionalität. Der Nutzer kann bei Belieben den aktuell aufgebauten Arbeitsplatz speichern und jederzeit einen existierenden Arbeitsplatz laden. Zudem bietet das System die Möglichkeit, im Hintergrund sämtliche Veränderungen innerhalb eines Arbeitsplatzes für den Nutzer unbemerkt regelmäßig zu persistieren - sofern der verwendete Arbeitsplatz bereits mindestens einmal gespeichert wurde und somit über eine definierte Bezeichnung verfügt. Dieses Speichern des Deltas geschieht jeweils genau dann, nachdem

mehr als vier Veränderungen im Arbeitsplatz registriert wurden. Eine Timerfunktionalität ist derzeit nicht implementiert.

Die Verwaltungsoberfläche bietet dem Nutzer umfangreiche Möglichkeiten, existierende Komponenten anzupassen und neue Komponenten hinzuzüfügen. Jede Komponente enthält Anpassungsfunktionalitäten sowohl für den Prozesscode, bzw. die Prozesscodes der Knoten, als auch für die funktional notwendigen Parameter. Diese können jederzeit hinzugefügt, angepasst und wieder entfernt werden.
Bislang fehlt für eine performantere Ausführung des Prozesses die Möglichkeit, diverse benötigte Libraries wie beispielsweise ein JDBC oder seprarate *Third-Party-Frameworks* an- oder abwählen zu können. So ist der Anwendungsserver zum aktuellen Stand stets in der Pflicht, sämtliche verfügbare Bibliotheken und Frameworks zu laden, unerheblich ob diese benötigt werden oder nicht.

6.2 Generizität der Komponenten

Die elementare Anforderung an das System *Mangold* war es, Komponenten weitestgehend generisch und modular aufbauen zu können. Der Nutzer soll in seiner Phantasie nicht von diversen technischen und logischen Hürden eingeschränkt werden, so dass er in der Lage sein kann, all jene ihm einfallende Arten von Komponenten zu erstellen.

Aufgrund der Verfügbarkeit des Prozesscode-Editors, mit welchem die Komponente auf beliebige Art konfiguriert und programmiert werden kann, ist diese Anforderung erfüllt. Aufgrund der umfangreichen Konfigurierbarkeit und Parametrisierung enthält der Prozesscode zudem die Möglichkeit, flexibel je nach aktuellem Gebrauch zu fungieren und kann dadurch, abhängig von den Anforderungen, sehr kurz und übersichtlich gehalten werden. Da als mögliche Prozesscode-Programmiersprachen etwa Groovy und Java verfügbar sind, ist hier demnach auch von einer Turing-vollständigen, universellen Programmierbarkeit auszugehen.

Da zudem Prozesscodes knotenpunkt- und nicht komponentenabhängig sind, der Administrator also unterscheiden kann zwischen Eingangs- und Ausgangsprozessen, verfügt er auch über die Möglichkeit, *Transformations*-Komponenten mit eingehenden und anschließend wieder ausgehenden Informationen zu erstellen. Somit kann er also auch flexibel beliebig lange Ketten möglich machen, ohne mittels diverser Workarounds arbeiten zu müssen.

Dank der Vereinheitlichung aller Komponenten auf das simple, plattformunabhängige Notationsmodell JSON wird sich im Übrigen von allgemeinen Plattformbeschränkungen freigemacht und eine flexible Erweiterbarkeit ge-

geben.

6.3 Nutzbarkeit

Die Bewertung der Nutzbarkeit einer Anwendung ist meist eher subjektiver Natur. Eine der wenigen Eigenschaften, welche definitiv Einfluss auf die allgemein empfundene Nutzbarkeit einer Anwendung haben und maximal objektiv ermittelt werden kann, ist jedoch die Performance der Anwendung. Für eine Analyse dieser wurde ein nichtlokaler Tomcat Application Server eingerichtet und auf diesem die Anwendung deployt. Diese wurde dann im Hinblick ihrer Reaktionsfähigkeit bei verschiedenen Mengen an Komponenten im Editor empirisch geprüft. Da sämtlicher Berechnungsaufwand jedoch auf Client-Seite, also im Browser des Nutzers geschieht, sind diese Ergebnisse stark abhängig von den Leistungsdaten und der Konfiguration des Systems des Anwenders.
Getestet wurde die Bedienbarkeit der Oberfläche bei 10, 25, 50 und 100 im Editor befindlichen Komponenten. Ein sogenanntes *Ruckeln* der Komponenten, also eine nicht-gleichmäßige Animation beim Verschieben einer Komponente, stellte sich auf dem getesteten System bei 25 Elementen im Editor ein. Dieses Ruckeln betraf jedoch bei 25 Elementen lediglich die Animation beim Verschieben der Komponenten. Animation etwa beim Erzeugen von Verbindungslinien waren hiervon nicht betroffen. Auch Animationen wie das Öffnen des Parameter-Editors einer Komponente lief bei 25 im Editor existierenden Komponenten unbehelligt performant.
Bei 50 im Editor bestehenden Komponenten war das Ruckeln dementsprechend verstärkt. Der Editor war subjektiv empfunden noch ausreichend bedienbar, wenn auch etwas störend verzögert. Eine Reaktion der Animation auf eine Mausbewegung beim Verschieben einer Komponente fand nach etwa 0,5 Sekunden statt. Auch andere Canvas-Animationen wie der Aufbau einer Verbindungslinie wirkten nun verzögert. Demgegenüber war jedoch einer Canvas-fernen Animation wie das Aufbauen des Property-Editor-Fensters komplett unbeeinflusst von diesem Effekt und baute sich weiterhin flott und ruckelfrei auf.
Wie zu erwarten, verschärfte sich dieser Effekt bei 100 Elementen deutlich. Eine Animationsreaktion auf eine Drag&Drop-Mausbewegung fand nach etwa 1,5 Sekunden statt und wirkte somit subjektiv unbedienbar. Nichtsdestotrotz blieb der Property-Editor weiterhin davon unabhängig performant.

Im Hinblick auf die allgemeine Bedienbarkeit und Übersichtlichkeit ist die subjektiv empfundene, übersichtliche Oberfläche zu erwähnen, welche funktional verwandte Elemente, wie die Navigation oder das Komponentenmenü, in ihren Bereichen gruppiert und dank etwa *Internationalization*-Funktionalität intuitiv und verständlich für den Endanwender zu bedienen ist. Zudem bieten sogenannte Notifications, also kleine Popups am rechten

unteren Bildschirmrand, bei Problemen oder wichtigen Informationen stets Einblick in eine eventuell fehlerhafte Bedienung oder aktuell stattfindende Prozesse. Der Nutzer des Editors wird zudem nicht belastet vom eigentlichen Prozess einer Komponente. Er erfährt lediglich den Namen dieser und ändert bei Bedarf ihre Parameter. Die eigentlichen Prozessinformationen in Form des Prozesscodes bleiben ihm verborgen und dem Administrator vorbehalten.
Hervorzuheben ist auch der *Default Field Configuration*-Editor in der Verwaltungsoberfläche, welcher dem Administrator die Erstellung und Bearbeitung von Parametern und Eigenschaften vereinfacht.
Optimierungsbedarf besteht dennoch weiterhin besonders in der Verwaltungsoberfläche. So ist speziell das Erstellen des Prozesscodes noch nicht ausreichend intuitiv, da dem Nutzer etwa allgemeine Informationen fehlen, auf welche Parameter er wie zugreifen kann und wie er sie für seine Zwecke verwenden kann. Ihm bleiben zudem jegliche Informationen verborgen, auf welche möglicherweise externen Bibliotheken er zudem zugreifen kann und wie und in welchem Format er verarbeitete Informationen letztlich zurückgeben kann und soll.

7 Fazit

7.1 Überblick

Innerhalb dieser Arbeit wurde ein Prototyp entwickelt, welcher mittels eines Drag&Drop-fähigen Editors verschiedenartige Komponenten einer Informationsverarbeitungskette aufnehmen und verarbeiten kann. Die Komponenten sind verknüpfbar und können Bestandteil einer beliebig langen Kette sein. Komponenten sind administrierbar und können dementsprechend mittels eines Prozess-Editors und separater Parameter beliebig erstellt und verwaltet werden. Der Administrator ist frei in der Erstellung und wird erwartungsgemäß nicht von diversen Einschränkungen blockiert.

Wie gezeigt wurden die Anforderungen an den Prototypen größtenteils erfüllt, wodurch ein weitestgehend leicht zu bedienender Editor erschaffen wurde, welcher mitunter Funktionalitäten für ein einfaches User-Management und eine dynamische Arbeitsplatz-Umgebung mit Persistenzfunktionalität bietet.

Einzelne Komponenten einer Informationsverarbeitungskette sind wie gezeigt aufgrund ihrer JSON-Basis und der vollwertigen Programmierbarkeit modular und generisch; der Ersteller einer Komponente kann ungehindert jede Art einer Quelle, Transformation oder Senke entwickeln und dem Endnutzer anbieten.

7.2 Ausblick

Wie im Abschnitt Evaluierung, Kapitel 6 gezeigt, weist das Projekt durchaus noch Lücken und Erweiterbarkeiten auf. So bietet es sich beispielsweise an, das Speichern der Workspace-Delta-Daten timerbasiert etwa alle zwei Minuten ausführen zu lassen. Ebenso sollte der Editor nur registrierten Nutzern zugänglich sein. Es sollte also eine Registrierungs-Oberfläche entworfen werden, in welcher neue Nutzer mit diversen zusätzlichen Daten wie Vor- und Nachname persistiert werden können. So müsste auch der Server verifizieren, ob die eingegebenen Login-Daten tatsächlich bereits in der Datenbank vorhanden sind oder nicht.

In Bezug auf die Komponentenverwaltung bietet es sich an, dem Nutzer eine Auswahlliste anzubieten, welche ihm darstellt, auf welche optionalen, zusätzlichen Bibliotheken und Frameworks er gegebenenfalls zugreifen kann. Diese sollte er bei Bedarf an- und abwählen können, wodurch der Server aussagekräftig Kenntnis darüber bekommt, welche zusätzlichen Pakete er einbinden muss. Dadurch wird sich die Prozesslast beim Ausführen der entworfenen Informationsverarbeitungskette merklich verringern.
Zudem ist es sinnvoll, zusätzlich zum Prozess-Editor-Feld dem Nutzer ein

Repertoire an fertig definierten Snippets für etwa Stringverarbeitung oder Listenmanipulation anzubieten und ihm eine Menge an nutzbarer Funktionen und Pakete aufzuzeigen. Zudem muss er eine Übersicht erhalten, wie er auf Parameter und sonstige Informationen einer Komponente zugreifen kann.

Da dieses Projekt lediglich den Editor für Verarbeitung von Informationsverarbeitungsketten umfasst, wird für eine volle Funktionalität dessen sowohl ein Persistenz-Server als auch ein Ausführungsserver benötigt. Beide sind im Rahmen eigener Studienprojekte bereits in Arbeit, bzw. bereits erarbeitet worden. Dementsprechend sind diese drei Projekte zusammenzuführen, indem dem Umfeld öffentliche Schnittstellen bekannt gemacht werden, auf welche dann zugegriffen werden kann.

Literatur

[1] Bootstrap - About. http://getbootstrap.com/about/.

[2] Das Phänomen World Wide Web. http://www.netplanet.org/geschichte/worldwideweb.shtml.

[3] DBPedia. http://de.dbpedia.org/.

[4] Google Mashup Editor. http://www.netzwelt.de/news/75964-google-mashup-editor-web-2-0-selbst-gemacht.html.

[5] Google Web Toolkit. http://www.fh-wedel.de/~si/seminare/ss13/Ausarbeitung/02.GWT/praesentation.pdf.

[6] Google Web Toolkit - Case Studies. http://www.gwtproject.org/casestudies.html.

[7] Google Web Toolkit Usage Statistics. http://trends.builtwith.com/framework/Google-Web-Toolkit.

[8] GWTProject - Debugging. http://www.gwtproject.org/doc/latest/tutorial/debug.html.

[9] GWTProject - Overview. http://www.gwtproject.org/overview.html.

[10] Kinetic JS. http://kineticjs.com/.

[11] Microsoft Popfly. http://en.wikipedia.org/wiki/Microsoft_Popfly.

[12] RDF Validator. http://www.w3.org/RDF/Validator/.

[13] The Future of GWT - Report. https://vaadin.com/documents/10187/b74a2275-232d-4d55-9158-fad1c8da5230.

[14] W3C. http://www.w3c.de/about/.

[15] W3C Semantic Web Activity. http://www.w3.org/2001/sw/.

[16] World Wide Web. http://info.cern.ch/hypertext/WWW/TheProject.html.

[17] What is an Ontology? http://userpages.uni-koblenz.de/~staab/Research/Publications/2009/handbookEdition2/what-is-an-ontology.pdf, 2009.

[18] Dean Allemang and Jim Hendler. *Semantic Web for the Working Ontologist*. Morgan Kaufmann Publishers, 2011.

[19] N. Bartelme. *Einführung in Geoinformation.* Springer-Verlag Berlin Heidelberg, 2005.

[20] Tim Berners-Lee, James Hendler, and Ora Lassila. *The Semantic Web*, pages 96 – 101. Scientific American, 2001.

[21] Marianne Buder, Werner Rehfeld, Thomas Seeger, and Dietmar Strauch. *Grundlagen der praktischen Information und Dokumentation: Ein Handbuch zur Einführung in die fachliche Informationsarbeit.* Walter de Gruyter, 1996.

[22] Robert T. Cooper and Charlie E. Collins. *GWT In Practice.* Manning Publications Co., 2008.

[23] Elena Demidova, Shirish Kucheria, Judith Plümer, Hartmut Polzer, Roland Schwänzl, Thomas Severiens, and Stefan Wenneker. Einführung in rdf. `http://www.iwi-iuk.org/seminarNotes/2/rdf-reader-v2.pdf`, 2006.

[24] Tom Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. `http://tomgruber.org/writing/onto-design.pdf`, 1993.

[25] Peter Haynes and Derrick Bowen. Houston Java Users Group - Google Web Toolkit. `http://de.slideshare.net/derrickbowen/gwt-hjug-presentation`.

[26] Tom Heath and Christian Bizer. *Linked Data - Evolving the Web into a Global Data Space.* Morgan & Claypool Publisher, 2011.

[27] Wolfgang Hesse. Ontologie(n). `http://lehre.ike.uni-stuttgart.de/wn/musoft/erg%C3%A4nzendetexte-gesamt/Ontologie.pdf`, 2002.

[28] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, and York Sure. *Semantic Web.* Springer Berlin Heidelberg, 2008.

[29] Rainer Kuhlen. Pragmatischer Mehrwert von Information. `http://www.kuhlen.name/MATERIALIEN/Texte/Pragmatischer_Mehrwert_von_Information.pdf`, 1989.

[30] Mozilla Org. Canvas Tutorial. `https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Canvas_tutorial`.

[31] Universität Saarland. Definition Information. `http://www.uni-saarland.de/campus/fakultaeten/fachrichtungen/philosophische-fakultaet-iii/fachrichtungen/informationswissenschaft/infowissthemen/datwissinf/definitioninformation.html`.

[32] Nigel Shadbolt, Wendy Hall, and Tim Berners-Lee. The semantic web revisited. http://eprints.soton.ac.uk/262614/1/Semantic_Web_Revisted.pdf, 2006.

[33] Adam Tacy, Robert Hanson, Jason Essington, Ian Bambury, and Christopher Ramsdale. *GWT In Action*. Manning Publications Co., 2 edition, 2011. Manning Early Access Program.

[34] Prof. Dr. Jörn von Lucke and Christian P. Geiger. Open Government Data - Frei verfügbare Daten des öffentlichen Sektors. http://www.zu.de/deutsch/lehrstuehle/ticc/TICC-101203-OpenGovernmentData-V1.pdf.

[35] W3C. Linked Data. http://www.w3.org/DesignIssues/LinkedData.html.

[36] W3C. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/.

[37] W3C. Resource Description Framework. http://www.w3.org/RDF/.

[38] W3C. Semantic Web and Other W3C Technologies to Watch. http://www.w3.org/2006/Talks/1023-sb-W3CTechSemWeb/W3CTechSemWeb.pdf.

[39] W3C. Semantic Web: Linked Data on the Web. http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#(24).

[40] W3C. The Canvas Element. http://www.w3.org/TR/html5/scripting-1.html#the-canvas-element.