UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4 Informatik: Arbeitsgruppe Staab/ISWeb

Graphpipes - Eine webbasierte graphische Benutzungschnittstelle (GUI) zur Definierung und Generierung von SPARQL-basierten Ansichten des Semantic Web

Studienarbeit

im Studiengang Computervisualistik

vorgelegt von

Thomas Winkler

Betreuer: Simon Schenk, Fachbereich 4, Arbeitsgruppe Stab / ISWeb
Erstgutachter:
Zweitgutachter:

Koblenz, im August 2008

Inhaltsverzeichnis

1 Einführung

Wie bereitet man komplizierte, technische Sachverhalte einfach und verständlich auf, damit sie auch der normalen Benutzer ohne tiefergehendes technisches Hintergrundwissen schnell und ohne lange Einarbeitungszeit und langwierige Erklärungen zu nutzen weiß?

In dieser Studenarbeit geht es um genau diese Frage – Nichtinformatikern die Vorzüge und der Arbeit mit semantischen (Such)anfragen zu erleichtern, wenn nicht so gar überhaupt erst zu ermöglichen, sowie die Neuentwicklung und SPARQL-Erweiterung *Networked Graphs*[Sch08] von Simon Schenk[1] innerhalb der AG Staab/Universität Koblenz[2] zu präsentieren.

> Networked RDF Graphs extend named graphs with a SPARQL based view mechanism. Briefly, a networked graph, among other statements, contains statements describing the graph itself in terms of SPARQL queries against other graphs. The networked graph then contains some explicitly listed content plus statements computed from the views on other graphs. *Simon Schenk, Networked Graphs [Sch08]*

Bestehenden Werkzeuge[3] sind nur für einen kleine Gruppe von Nutzern im Forschungsbereich gedacht und dementsprechend bieten sie weder eine ansprechende und einfache Benutzungschnittstelle.

Vorrausgehend an die Umsetzung der Software stand die Überlegung, welche Grundvorraussetzungen sowohl in der Auswahl des Entwicklungsmediums sowie der graphischen Darstellung erfüllt sein müssen, um einen möglichst großen Interessentenkreis anzusprechen als auch ein hohes Maß an Nutzerfreundlichkeit zu bieten. Jakob Nielsen[4] stellt diese Vorüberlegungen zur Konzeptionsphase in einem Interview mit InformationWeek als wichtigen zentralen Punkt für die Akzeptanz der Anwendung durch den Nutzer dar:

> The Web puts user experience of the site first [..]. On the Web, users first experience the usability of a site and then buy something. Give users a good experience and they're apt to turn into frequent and loyal customers. But the Web also offers low switching costs; it's easy to turn to another supplier in the face of even a minor hiccup. Only if a site is extremely easy to use will anybody bother staying around. *Jakob Nielsen and Donald A. Norman, Informationweek [JN08]*

Anschließend wurde, wie in Kapitel 6.5.1 zu sehen, aufgrund der vorherigen Überlegungen entschieden, mit welchem Systemaufbau diese Vorgaben erfüllt werden können.

---

[1] Simon Schenk: http://www.uni-koblenz.de/ sschenk/

[2] Universität Koblenz-Landau: http://isweb.uni-koblenz.de/

[3] Beispielsweis http://www.daml.org/cgi-bin/dumpont?http://www.daml.org/2001/10/html/airport-ont

[4] Jakob Nielsen: http://www.useit.com/jakob/

2 Graphpipes

Das Semantic Web ist die Erweiterung beziehungsweise Neuinterpretation des bestehenden WorldWideWebs um die Möglichkeit, nicht nur auf die Daten an sich, sondern auch auf die Verknüpfungen und Beziehungen zwischen diesen Information zuzugreifen und damit auch neues Wissen und neue Informationsebenen zu generieren. Das Semantic Web bietet diese Möglichkeit sowohl für den Menschen als auch insbesondere in maschinenlesbarer Form.

> The most widely accepted defining feature of the Semantic Web is machine-usable content.“ *Michael Uschold, Where are the Semantics in the Semantic Web? [Usc01]*

Tim Berners-Lee[5] fügt in seinem Essay *Semantic Web Road map*[BL98] hinzu, dass das Semantic Web ein Datennetz ist, und in vielen Dingen wie eine globale Datenbank arbeitet.

> The Semantic Web is a web of data, in some ways like a global database. The rationale for creating such an infrastructure is given elsewhere [Web future talks c] here I only outline the architecture as I see it.” *Tim Berners-Lee, aus Semantic Web Road map [BL98]*

Als Werkzeug steht mit RDF[6] ein vom W3C[7] spezifizierter Standart für die Definition der Graphbeziehunen und mit SPARQL[8] ein Absprachefrage für die Inferenz dieser Beziehungsnetze zur Verfügung.

Tim Bernes Lee spricht in diesem Zusammenhang von einem *Netz der Beziehungen*:

> If HTML and the Web made all the online documents look like one huge book, RDF, schema, and inference languages will make all the data in the world look like one huge database. *Tim Berners-Lee, aus Weaving the Web[BL99]*

[5]Sir Tim Berners-Lee: http://www.w3.org/People/Berners-Lee/
[6]RDF: Resource Description Framework
[7]W3C: World Wide Web Consortium, http://w3.org
[8]SPARQL: SPARQL Query Language for RDF [W3C08], http://www.w3.org/TR/rdf-sparql-query/

2.1 Zielsetzung und Motivation

Problem 1: Betrachtet man die Oberflächenwelt des Internets so fällt auf, dass der Entwicklung des Semantic Webs nicht viel Aufmerksamkeit geschenkt wird und auch bestehenden Anwendungen nicht die Resonanz außerhalb der Entwicklergemeinde erzeugen können, die man erwarten würde.
Nigel Shadbolt und Wendy Hall beschreiben dieses Problem in 'The Semantic Web Revisited':

> This simple idea, however, remains largely unrealized. A Web of data and information would look very different from the Web we experience today.
> *Nigel, Shadbolt: [NS06]*

Bestehende Arbeiten decken meist nur einen ganz speziellen Anwendungsfall ab und sind auch in ihrer Präsentation eher einfach, wenn nicht sogar schlicht gehalten. Es ist aber schwer, diese Werkzeuge dem Normalanwender in die Hand zu geben, das sie direkt auf der Basis des Problems ansetzen und eine gewisse Grundbildung in der Problemdomäne vorrausgesetzt wird. Es ist fast so, als würde man von einem Autofahrer verlangen, dass er den Motor seines Autos selbst zündet und das Benzingmisch per Schalter einstellt.
Russell B. Williams, Professor an der Hong Kong Baptist University vergleicht die Problematik des Verstehens eines Problembereiches mit der der Art wie sich Menschen auf dem Wasser fortbewegen in seinem Essay *Web Surfing ? How about a new analogy for experiencing the Web?Part 2)* [Wil03]

> These [..] types of interaction with the Web tell us a great deal about the people and purposes brought to the ocean of information, applications and experiences. Knowing which kind of person is in the water at your site will enable you to design applications, content and experiences that serve that individual's activities and needs. It is possible to look at the three types of users along a hierarchy of involvement. Swimmers are looking for a place to splash around and relax for a while. A snorkeler is interested in more activity and higher involvement but can still be attracted or distracted by style and functionality. These are the individuals who are looking for something and it is up to the designer to show them that what is available in this particular spot is something that has value for them. Meanwhile, the scuba divers are highly focused and motivated, they are on a mission, and a site can be useful or not. The challenge for designers is communicating to scuba divers what is possible to do in this particular part of the ocean. *Prof. Russell B. Williams, [Wil03]*

Ziel 1: Es soll eine ansprechenende, einfache Oberfläche geschaffen werden, die die Komplexität von SPARQL verdeckt und intuivie Metaphern zur Veranschaulichung und arbeitserleichterung zur Verfügung stellt.

Problem 2: RDF kann als Datenmodell keine (rekursiven) Referenzen auf andere Datenbestände abdecken. Sollen Informationsstrukturen aus anderen RDF-Graphen eingebunden werden, so müssen die komplett in den Graphen kopiert werden. Werden am ursprünglichen Graphen Änderungen vorgenommen, so müssen diese per Hand abgeglichen werden.

> The semantic web suffers from a lack of user generated content and from a lack of services to kick-start wide adop- tion. One of the reasons for these deciencies is a missing mechanism for easy, distributed data reuse and integration: As we are talking about quite large datasets in the exam- ples listed above, already the necessary infrastructure for holding the resulting amounts of data keeps many people from oering such services. In order to encourage people to build new services based on existing data, a more exible and distributed mechanism is needed. *Simon Schenk, aus Networked Graphs [Sch08]*

Simon Schenk nimmt sich diesem Problems in seinem Paper *Networked Graphs: A Declarative Mechanism for SPARQL Rules, SPARQL Views and RDF Data Integration on the Web*[Sch08] an und schlägt eine Syntaxerweiterung von RDF vor, die es mit eingebetteter SPARQL-Syntax erlaubt, beliebig viele Graphen bzw. Anfragen an Graphen zu referenzieren.
Ziel 2: Graphpipes soll ein Beispielanwendung für *Networked Graphs* sein

2.2 Darstellung des Lösungsansatzes

Graphpipes ist sowohl eine webbasierte, im Browser laufende Anwendung, als auch eine serverseitige Plattform, die die Kommunikation zwischen den Clients der Nutzer, der Datenbank und Sesame[9] als sematische Inferenzbasis bereitstellt.

Zentrales Thema der Präsentation auf Clientseite ist der Fluß, die Veränderung und Filterung von Informationen. Hierfür wurde als Metapher eine abgewandelte Form eines Flussdiagramms ausgesucht: Die Benutzungsschnittstellt lehnt sich an Yahoo! Pipes an. Yahoo! Pipes beschreibt sich als Werkzeug, um Daten zu kombinieren, sortieren und filtern.

> Like Unix pipes, simple commands can be combined together to create output that meets your needs: combine many feeds into one, then sort, filter and translate it. *Yahoo! Pipes [htt08a]*

Es besteht nicht der Anspruch sämtlich Funktionen und Möglichkeiten von SPARQL in Form von graphischen Mitteln abzubilden, der Augenmerk liegt auf den grundlegenden Funktionen und der Heranführung und Umsetzung des Problembereiches in graphischer Form.

[9] Sesame is an Opensource RDF-Framework with support for RDF-Schema inferencing and querying. Vgl [Ope]

Graphpipes wurde bewußt webbasiert entwickelt, um einer möglichst breiten Interessenschicht Zugang zu ermöglichen und sich nicht auf einer Rechner- oder Betriebssystemplattform zu begrenzen.

3 Semantic Web

Betrachtet man das WorldWideWeb in seiner aktuellen Form, so fällt auf, dass die meisten Informationen für die Interaktion mit Menschen aufbereitet werden. Zusammenhänge zwischen den Daten müssen aufwändig erstellt werden. Die Maschine ist nur zweite Wahl, wenn es um Informationen geht. Um genau diese Problem zu lösen und eine Basis für ein Semantic Web zu legen, wurden Spezifikationen wie RDF und SPARQL entwickelt.

3.1 SPARQL

SPARQL ist ein rekursives Akronym für *SPARQL Protocol and RDF Query Language*. Sie ermöglicht ein Patternmatching auf Informationen in RDF-Graphen.

3.2 RDF

RDF is a directed, labeled graph data format for representing information in the Web. RDF is often used [...] to provide a means of integration over disparate sources of information. (vgl. http://www.w3.org/TR/rdf-sparql-query/[rdf])

3.3 Networked Graphs

Networked Graphs sind eine Erweiterung von Named Graphs (siehe[htt]). Es können Ansichten (Views) mit SPARQL definiert werden, so dass die Graphen neben den RDF-definitionen Abfragen auf andere RDFgraphen enthalten können.

> Named Graphs is the idea that having multiple RDF graphs in a single document/repository and naming them with URIs provides useful additional functionality built on top of the RDF Recommendations." *W3C [htt]*

So müssen Daten nicht mehr an einer vielzahl von Orten gepflegt werden, sondern nur noch an einer Stelle - sämtliche Graphen, die sich auf diese Daten beziehen werden mitaktualisiert, da die Daten nicht in den Graphen kopiert werden, sondern nur die Beziehung definiert wird, welche dann zur Laufzeit evaluiert wird.

Simon Schenk beschreibt unter anderem folgende Vorzüge dieses Vorschlages in [Sch08]:

- Informationswiederverwendung
- Integration von 3rd-party Daten
- Rekursion

4 Situationsanalyse

Dieses Kapitel wird sich mit den der Arbeit vorrausgehenden Überlegungen beschäftigen, wie man die Problemstellung grapisch und inhaltlich aufbereiten soll. WelcheVorraussetzungen bestehen oder müssen im Weiteren bestehen?

4.1 Anforderungen

Wie zuvor erwähnt sollen einige Eckpunkte während der Programmierung beachtet werden. Die Software soll

- einfach und ohne Probleme für einen breiten Nutzerkreis zu bedienen sein.
- direkt und unmittelbar zu erreichen sein.
- nicht an proprietäre technik gebunden sein.
- optisch ansprechend gestaltet werden.
- dem Nutzer über eine ensprechende Methaper die Grundfunktionalität von SPARQL und Networked Graphs zur Verfügung stellen.

Zusätzlich wäre es hilfreich und wünschenswert, wenn eine Hilfe und Erklärungsbeispiele mitgeliefert würden, sowie die Software auf einer vorgeschalteten Internetseite mit Screenshots präsentiert werden könnte, um die initiale Hürde vor der ersten Nutzung so gering wie möglich zu halten.

Der Verzicht auf im WWW verfügbare und verbreitete, aber proprietäre Technologien ist beabsichtigt, auch wenn beispielsweise Adobe Flash eine technisch geeignete und mit 99% Marktverbreitung[10] auch verfügbare Plattform wäre, da gerade Graphpipes als Beispielanwendung für eine auf OpenSource-Software basierende Anwendung offen zur Verfügung gestellt werden soll. Aus dem selben Grund wird Graphpipes auch unter der MIT-Lizenz veröffentlicht, die die Software an sich frei zum allgemeinen Nutzen zu Verfügung stellt und auch die Veränderung und Weiterverbreitung ausdrücklich gestattet.

In Kapitel 5 (Pflichtenheft) werde ich genauer auf die gewünschten Eigenschaften der Software eingehen.

4.2 Zielgruppenanalyse

Wie bereits erwähnt, soll Graphpipes dem allgemeinen Publikum bereitgestellt werden, sowie desweiteren als Präsentations- und Vorführobjekt bei Vorträgen für Networked Graphs dienen.

Als Konsequenz folgt, dass auf Shell- bzw. Texteingaben weitestgehend verzichtet und auf die graphische Representation besonderen Wert gelegt werden soll. Auch muss die Nutzerführung logisch konzipiert werden, damit keine Situationen in der Nutzung

[10] Markverbreitung Adobe Flash liegt nach eigenen Angaben von Adobe, Inc. bei 99% - siehe auch [Abo]

des Systems entstehen, die den Nutzer dazu veranlassen könnten, den Gesamtüberblick zu verlieren. Genauer werde ich in Kapitel 6 auf diese Problematik, insbesondere der Interaktionsarten mit dem Nutzer in Form von Informationseingaben eingehen: Wie agiert der Nutzer am effektivsten mit Graphpipes ohne im Arbeitsfluß gestört zu werden?

5 Pflichtenheft

Folgende Anforderungen an die Systembeststandteile von Graphpipes werden in Bezug auf die Architektur und die Benutzungsschnittsstelle gestellt.

5.1 Allgemeine Vorraussetzungen

- Graphpipes soll per Internet erreichbar sein
- Die Software soll mit einem standard-kompatiblen Webbrowser (Firefox) benutzbar sein
- Die Software soll modular aufgebaut sein (multi-tier)
- Es soll möglich sein, ggf. ein weiteres Frontend aufzusetzen
- Die Kommunikation des Frontendes und der Backendserveranwendung geschieht per REST-Interface

5.2 Gestaltung

- Die GUI soll sich an den Interfaceelementen von Desktopbetriebssystemen anlehnen
- Das Interface soll in sich einheitlich gestaltet sein
- Es sollen verständliche Fehlermeldungen insbesondere bei Formulareingaben ausgegeben werden
- Alle Interaktionen (Pipes) mit dem Server werden gecached.

5.3 Anwendungbestandteile

5.3.1 Erklärungsseite

- Die Grundlagen von Graphpipes sollen erklärt werden
- Der Nutzer soll sich anmelden können
- Der Nutzer soll sich registrieren können
- Es sollen Screenshots zur Veranschaulichung gezeigt werden
- Es muss ein Impressum vorhanden sein
- Die Anmeldung soll innerhalb der Webseite möglich sein ohne auf einen neue Seite zu springen
- Die Registrierung soll auf einer eigenen Seite ablaufen

5.3.2 Login

- Der Nutzer muss seinen Namen, seine Email und sein Passwort hinterlegen
- Nach der erfolgreichen Registrierung wird der Nutzer direkt eingeloggt
- Bei fehlerhaften Angaben werden die Fehler und das Formular angezeigt

5.3.3 Ladingpage in Graphpipes (Dashboard)

- Die Nutzer sollen neue Pipes erstellen können
- Die Nutzer sollen ihre Pipes löschen können
- Die Nutzer sollen ihre Pipes editieren können
- Die Nutzer sollen ihre Pipes öffnen/ansehen können
- Die Nutzer sollen RDF-Graphen definieren können
- RDF-Graphen haben einen Namen und eine URI
- Jedem Nutzer besitzt einen öffentliche Seite, auf der seine Daten beschrieben sind (Account page)

5.3.4 Ansicht einer Pipe

- Es gibt eine oberste, allgemeine Navigationsebene
- Es gibt eine Arbeitsfläche
- Es gibt eine Werkzeugleiste mit SPARQL-Operationen
- Die Navigation enthält folgende Operationen
  - Den Arbeitsbereich verlassen und zur Landingpage wechseln
  - Die Pipe löschen
  - Eine neue Pipe erstellen
  - Einen RDF-graphen ohne Interaktionsstörung hinzufügen
  - Die Werkzeugleiste ein- und ausblenden
  - Die Nutzerdaten ändern
  - Graphpipes verlassen
- Es gibt in der Pipeansicht Pfadangaben
- Pfadangaben stellen die Tiefenstruktur der verschachtelten Pipes dar
- Die Pipe kann evaluiert/ausgeführt werden
- Die erzeugte Query kann angezeigt werden, ohne den Server anzusprechen
- Alle Operationen im Arbeitsbereich sind ohne Interaktionsstörung

5.3.5 Arbeitsfläche

- Jede Veränderung im Frontend wird im Backend gespeichrt
- Die Arbeitsfläche soll sich nach einem Reload wieder exakt gleich aufbauen
- Die SPARQL-Operationen sollen durch Darstellung als Kästen eigenständig erkennbar sein (Nodes)
- Die Operationen können durch Kanten (Edges) verbunden werden
- Nodes können gelöscht werden
- Nodes können verschoben werden
- Die Kanten haben eine Richtung
- Jede Kante wird an einem Endpunkt verankert (Terminal)
- Jedes Terminal kann nur eine Kante annehmen
- Kanten können wieder gelöscht werden
- Es gibt eine "Endnode", die das Ende der Pipe definiert
- Folgende Operationen werden definiert
  - Neuen RDF-Graphen hinzufügen
  - Einen Subgraphen (Subpipe) hinzufügen
  - Conditionals (Where)
  - Filter
  - Contruct für Networked Graph
  - Select für RDFgraphen
  - Es kann eine Sonderoperation "!=" genutzt werden

6 Architektur

Der Aufbau der Anwendung gliedert sich in zwei Hauptbestandteile. Zum einen die graphische Gestaltung und zum anderen die technische Architektur. Die Gestaltung bestimmt das Verhalten des Frontends im Browser und legt die Interaktion mit dem Nutzer fest, die technische Komponente beschreibt die Interaktion zwischen Browser, Sesame als Sematische Datenbank und der Businesslogik im Backend.

6.1 Gestaltung

In Hinblick auf die Gestaltung müssen, wie bereits zu Anfang beschrieben, verschiedene Aspekte beachtet werden, um die Erwartungshaltung der Nutzer an eine Desktopapplikation zu erfüllen: Graphpipes bietet innerhalb der Arbeitsfläche einen unterbrechnungsfreien Workflow, es werden keine Unterseiten aufgerufen und alle Änderungen per AJAX an den Server geschickt. An dieser Stelle verhält sich Graphpipes wie eine Desktopanwendung. Diesem Umstand wird auch in der Gestaltung der Schaltflächen, Dialogboxen und Fenster Rechnung getragen: Sie entsprechen in Form, Funktionsweise den bekannten Elementen von Desktopprogrammen.

Anderseits findet der Nutzer Graphpipes per Webbrowser und erwartet im Medium Internet einige Besonderheiten, die es bei nativen Anwendungen nicht gibt: Jede Seite und jeder Zustand müssen jederzeit wiedererreichbar sein, d.h. der Nutzer muss in der Lage sein, Links zu speichern und zu versenden. Jede interaktion beeinhaltet einen eingenständigen Statewechsel auf Seiten des Backends. HTTP ist zustandlos und so kann nicht davon ausgangen werden, dass eventuel für Interaktionen notwendige Vorraussetzungen immer gegeben sind. Eine weitere Besonderheit ist, dass es im Medium Browser keine Doppelklicks gibt. Sämtliche Interaktionen müssen demnach mit einem einfachen Klick bzw. direktem Drag-and-Drop ausgeführt werden können.

Die Anwendung präsentiert sich je nach Nutzerstatus – ob eingeloggt oder nicht – entweder als Erklärungsseite oder als Werkzeug in der Arbeitsansicht.

Abbildung 1: Frontansicht mit Erklärungstext

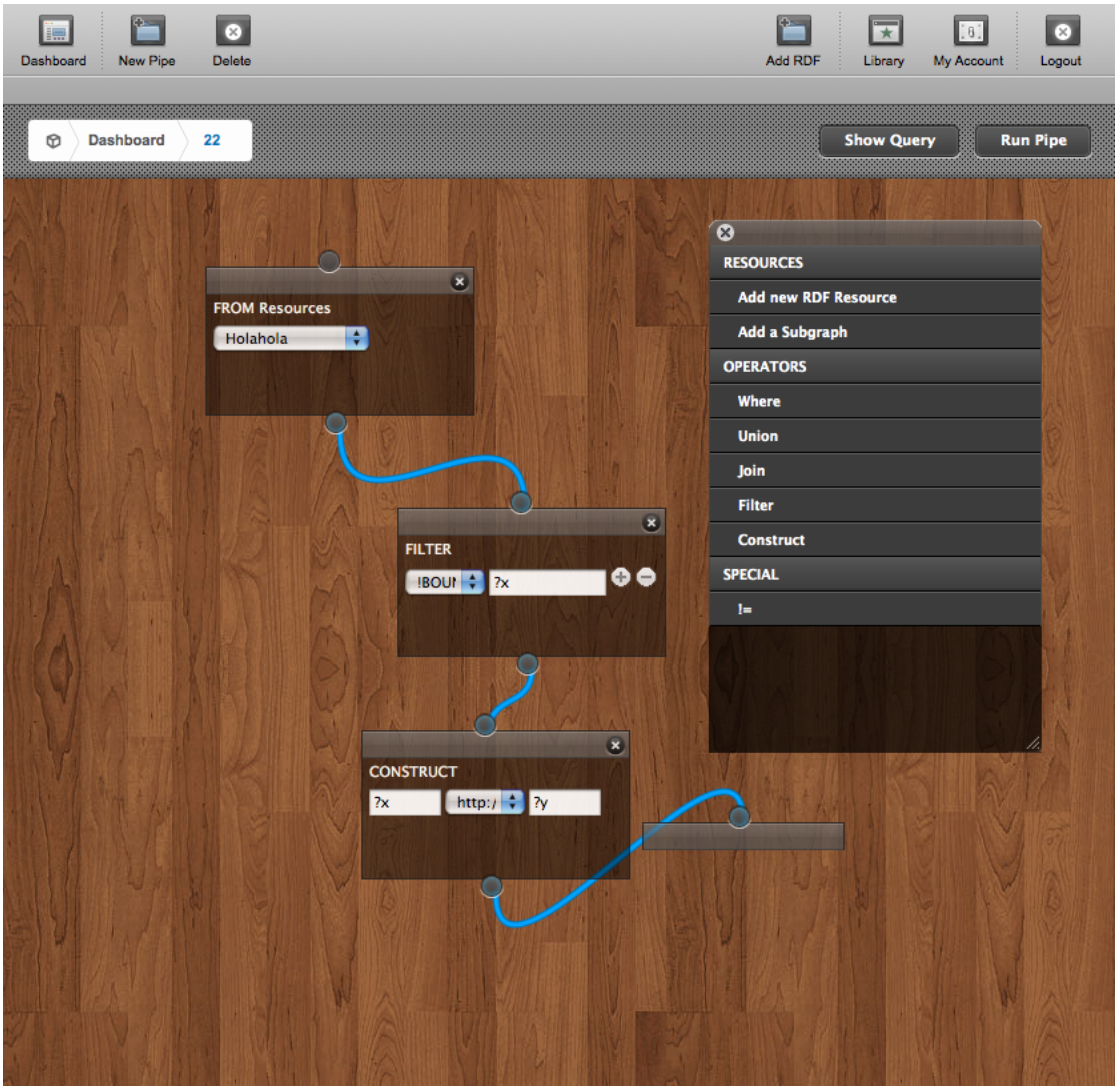

Abbildung 2: Ansicht der Arbeitsfläche

Die in der Gestaltung (siehe Abb. 1, 2) verwendenten Metaphern sind neben der Arbeitsfläche, dem Desktop, der durch die Holzelemente nochmals gestalterisch näher an den reelen Schreibtisch geführt wird, auch die Informationsflußröhern (Abb. 2) – die Pipes – die die Informationen von oben nach unten in die Sammelnodes fließen lassen. Beide Sinnbilder sollen dazu dienen das abstrakte Konzept der Anfrageprozedur und der komplizierten Technik begreiflicher zu machen.

6.2 Styleguide und Interfacebestandteile

Im Vorfeld habe ich alle Elemente der Benutzungsschnittstelle (Buttons, Navigationsleiste, Nodes, Kanten, Dialogfelder etc.) entworfen, um ein einheitliches Erscheinungsbild gestalten zu können und die beiden in Interaktionsmodelle und ihren jeweiligen Anwendungszweck definiert. Wichtig war mir, dass an jeder Stelle darauf geachtet wird, das sowohl Interaktionsmodell als auch Erscheinungsbild der Nutzererwartung entsprechen

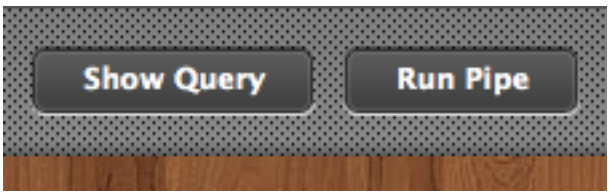

Abbildung 3: Graphpipes UI Elemente (Buttons)

6.3 Interaktionsmodelle

Innerhalb der Anwendung habe ich zwei Interaktionsmodelle definiert, die mit den zuvor beschriebenen verwendeten Interaktionsmetaphern von Desktopapplikationen und Webseiten übereinstimmen.

Die gestaffelte Interaktion entspricht der Navigation innerhalb einer Webseite. Nach der Bestätigung des Nutzers per Klick oder Reload wird die gesamte Seite neu aufgebaut und präsentiert (Pagereload). In dieser Navigationsart ist es bei Eingaben sinnvoll, gegebenfalls nicht das gesamte Interface anzuzeigen, um den Nutzer auf diesen einen Interaktionsschritt zu fokusieren. So verlässt der Nutzer bei der Registrierung die Erklärungsseite samt Hauptnavigation (Abb. 4) und bekomme eine eigenständiges Formular angezeigt. So kann er sich auf seinen Aufgaben konzentrieren und wird nicht von ablenkenden Informationen gestört.[11]

[11] vgl. *Defensive Design for the Web*, Seiten 140ff. [ML04]

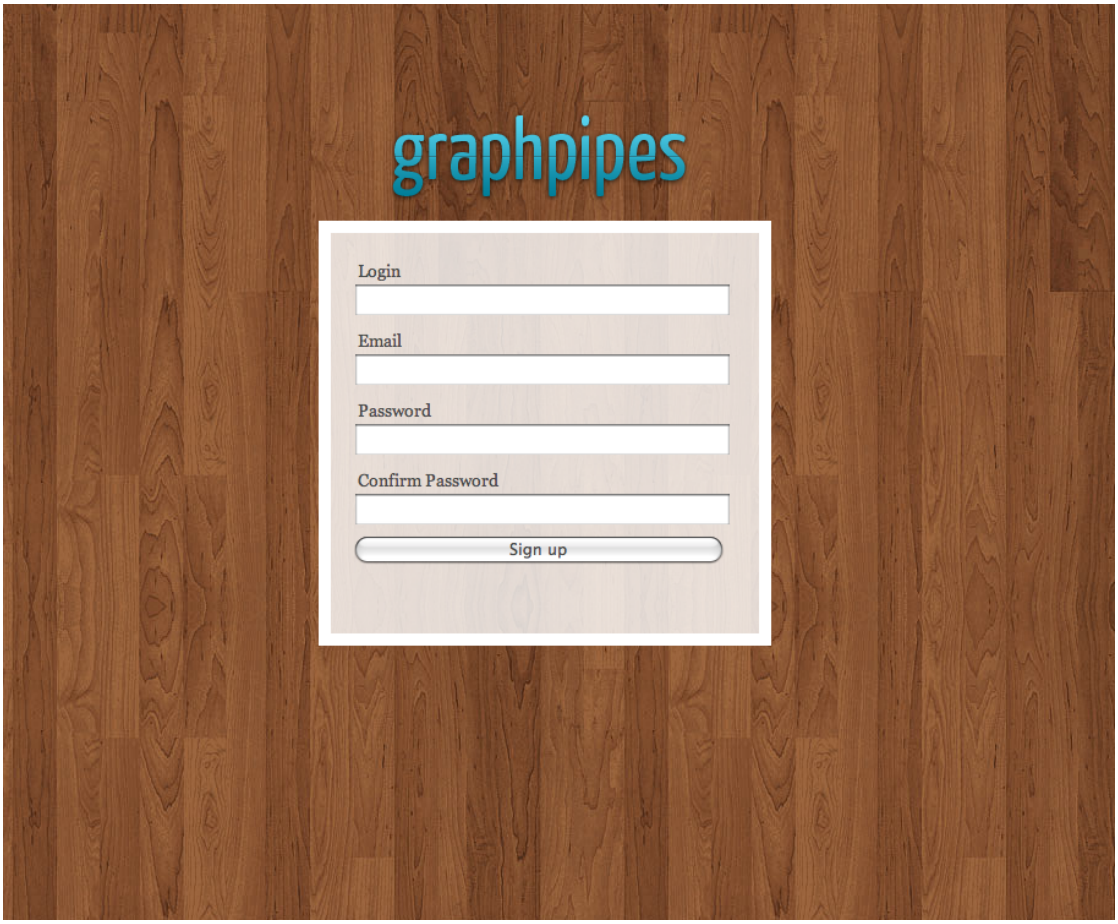

Abbildung 4: Anmeldebildschirm. Es werden keine weiteren Seitelemente angezeigt.

Die parallele Interaktion entspricht der des Windows/Mac-Desktops. Änderungen werden im Hintergrund per AJAX an den Server übermittelt, die Elemente auf der Seite per Javascript gesteuert; es gibt keinen Pagerefresh. Soll der Nutzer hier Angaben machen, wird ihm ein modaler Dialog oder Sheet (Abb. 5) gezeigt. Sollte er diesen nicht ausfüllen und den Vorgang abbrechen, hat dies keinen Einfluß auf den vorherigen und kommenden Zustand der Anwendung und auch bei erfolgreichem Abschluss bleibt der Nutzer auf dieser einen Seite. Wichtig ist dies in erster Linien um ein freies und einfaches Arbeiten zu ermöglichen.

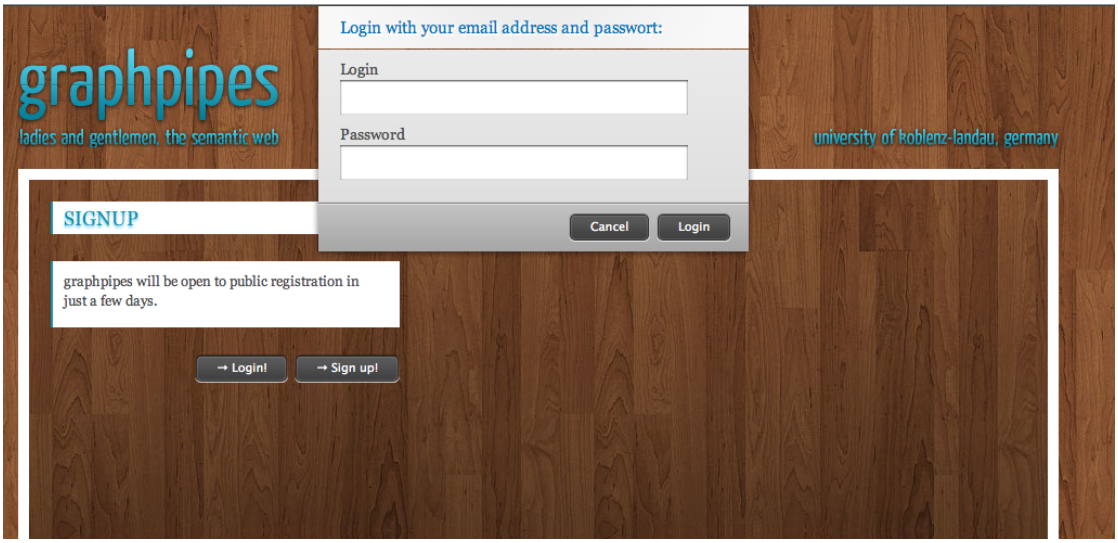

Abbildung 5: Anmeldebildschirm. Der Seitenkontext bleibt erhalten.

6.4 Nutzerpfade

Ich möchte im Folgenden kurz und exemplarisch einen Interaktionspfad und die damit verbundenen Entscheidungen, die GUI zu gestalten, darlegen.

Erreicht ein Nutzer zum ersten Mal die Seite ist es wichtig, dass er unmittelbar einen Eindruck davon gewinnt, was in erwartet, denn ohne Grunderwartungshaltung und -verständnis von Graphpipes wird die Anwendung für den Nutzer nicht funktionieren. Der Signup-button befindet sich aus diesem Grund auf der letzten der Präsentationsfolien (Abb. 7) - der Loginbutton über den sich Bestandsnutzer einloggen können ist aber schon auf der ersten Folie schnell erreichbar. (Abb. 6)

Abbildung 6: Frontansicht mit Erklärungstext, 1. Folie

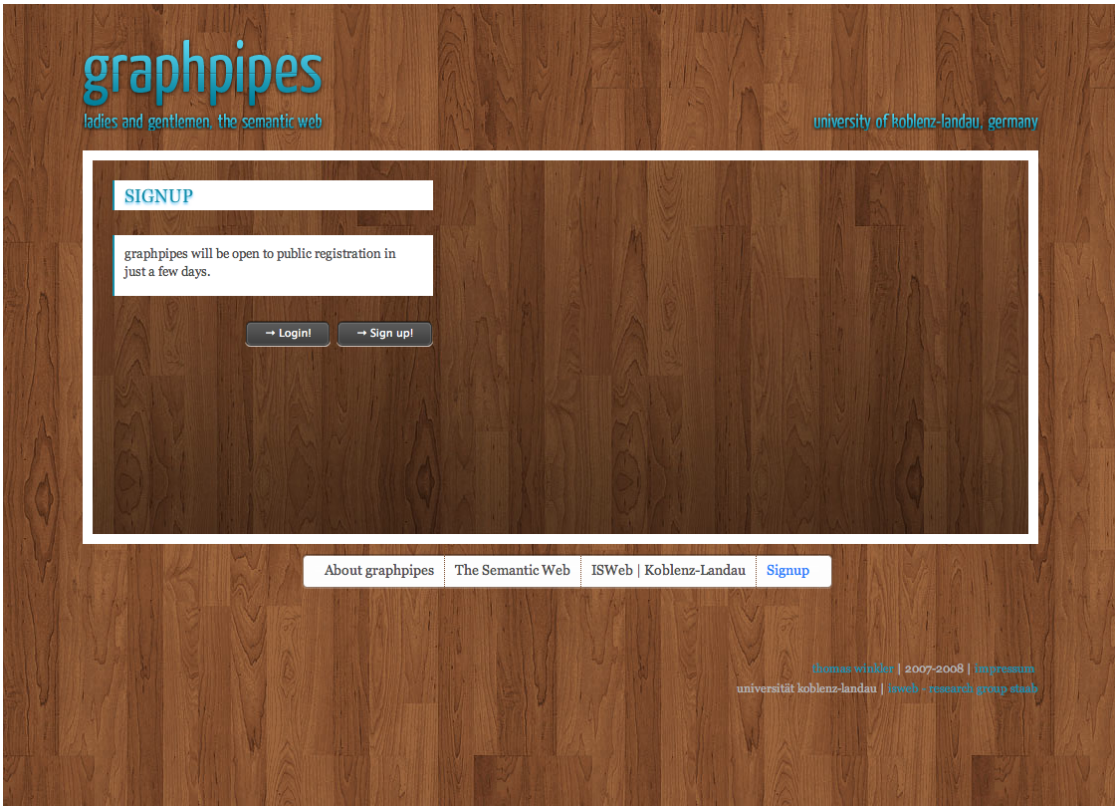

Abbildung 7: Frontansicht mit Erklärungstext, letzte Folie

6.5 Technik

Graphpipes ist in zwei Hauptbestandteile aufgespalten: Clientseitig in HTML/CSS und Javascript programmiert, wird serverseitig Ruby On Rails eingesetzt. Beide Teile machen von frei verfügbaren weiteren Frameworks gebrauch. In Kapitel /refsec:Programmierumgebung wird die verwendete Technik erläutert und die allgemeine Struktur der Anwendungsbestandteile sowie ihre Interaktion untereinander beleuchtet.

6.5.1 Programmierumgebung

Für die serverseitige Entwicklung wurde das Opensource-Webframework RubyOnRails[12] gewähnt. RubyOnRails (RoR) ist in Ruby[13] implementiert, einer objektorienierten Programmiersprache, die sich stark an Smalltalk[14] und Perl[15] anlehnt. RoR stellt eine komplette Umgebung für die Erstellung von Model-View-Controller Webanwendungen zur Verfügung. Zudem wird mit ActiveRecord ein ORM-Adapter ("Object Relational Mapping) bereitgestellt, das das von Fowler[16] in *Patterns of Enterprise Application Architecture* [Fow02] beschrieben *Active Record Pattern* umsetzt. Die Anbindung an SQL-basierte Datenbanken wird so erleichtert und abstrahiert.

> Rails is a full-stack framework for developing database-backed web applications according to the Model-View-Control pattern. From the Ajax in the view, to the request and response in the controller, to the domain model wrapping the database, Rails gives you a pure-Ruby development environment. To go live, all you need to add is a database and a web server. *RubyOnRails Website [htt08b]*

Der Kontrollfluß innerhalb von RoR stellt sich nach eingegangenem Request folgendermaßen dar:

Der Request wird über einen Routingmechnismus analysiert und über die zuvor vom Entwickler definierten URL-Objekt-Zuordnungen an die entsprechenden Controllerobjekte übergeben.

Diese koordinieren die Verbindung mit der Datenbank, die Datenaufbereitung und -transformation, in Fall von Graphpipes auch die Kommunikation mit Sesame per REST-Interface, und rendern die Views, die je nach im Request definiertem Accept Header entsprechend ausgegeben werden. So kann ein Controller mit der beinhalteten Buisnesslogik mehrer verschiedene Anfragetypen bearbeiten. Dies stellte sich in der Verlauf der Umsetzung von Graphpipes als sehr vorteilhaft heraus, da ja nach Interaktionsmodell (vgl. oben) entweder HTML oder Javascript (AJAX) als Rückgabewert zur Verfügung stehen musste, die Logik aber nur einmal implementiert werden musste.

[12] Ruby On Rails ist ein Fullstack-Webframework, http://www.rubyonrails.org [htt08b]
[13] Die Ruby Programmiersprache, http://www.ruby-lang.org
[14] Die Smalltalk Programmiersprache, http://www.smalltalk.org/smalltalk/whatissmalltalk.html
[15] Die Perl Programmiersprache, http://www.perl.org/about.html
[16] Martin Fowloer, Thoughtworks, http://martinfowler.com/aboutMe.html

Das Frontend wird aus der gleichen Anwendung heraus generiert, es ist aber auch ohne weiteres möglich, eine komplett eigenständige Applikation anzubinden. Genauer werde ich auf diesen Punkt in 5.2.3.3 (REST) eingehen.

Muss für den ersten Request noch das gesamte Interface an den Client verschickt und dort per Javascript aufgebaut werden, so geschehen die weiteren Anfragen im Hintergrund. Das Frontend macht sich hier die enge Verzahnnung mit dem RoR-Backend zu Nutze und geht einge Abkürzungen: So ist der Javascript Quellcode an vielen Stellen nicht statisch angelegt und wird in immer der gleichen Form an den Browser geschickt, sondern wird vor Auslieferung dynamisch verändert, um dem jeweiligen State des Clients besser zu entsprechen: Wird bespielsweise eine Anfrage an die 3. Pipe des Nutzers mit der Nummer 66 gestellt, so wird innerhalb der HTML Seite nicht die Datei /javascripts/stage.js eingebunden, sondern die gleichen Anfrageoperatoren für die Javascripttransformation ein weiteres Mal durch den Rails-Stack (vgl. Abschnitt 6.5.1) geschickt:
Aufgerufen wird `/javascripts/stage/66/3.js`

Auf diese Art kann die Ajax-Javascript-Response Informationen aus dem Backend transportierten.
Beispiel (`app/views/javascripts/nodeBox.js.erb`):
So wird aus

```
nodeBox.addModule = function (element, namespace) {
  var user_id = <%= params[:user_id] %>
  var pipe_id = <%= params[:pipe_id] %>
[..]
}
```

browserseitig

```
nodeBox.addModule = function (element, namespace) {
 var user_id = 66
  var pipe_id = 3
}
```

6.5.2 Zusammenfassung der Aufgabenbestandteile (Systemaufbau)

Im Folgenden möchte ich die Struktur der Bestandteile von Graphpipes genauer beschreiben, zuvor aber einen Gesamtüberblick liefern. Wie in Figur 8 zu sehen, interagiert der Nutzer(Client) in erster Linie mit dem Browserbasierten Frontend. Die Kommunikation zwischen Frontend und Backend erfolgt per HTTP auf Basis der REST-Spezifikationen. Das Backend wiederum koordiniert die Verteilung, Speicherung und Weiterleitung der Daten an die Datenbank, Sesame als SPARQL-Abfragerepository und ggf. den Workerthreads, die für um langandauerende Operationen gestartet werden. Auf das Backend kann auch per REST-API direkt zugegriffen werden.

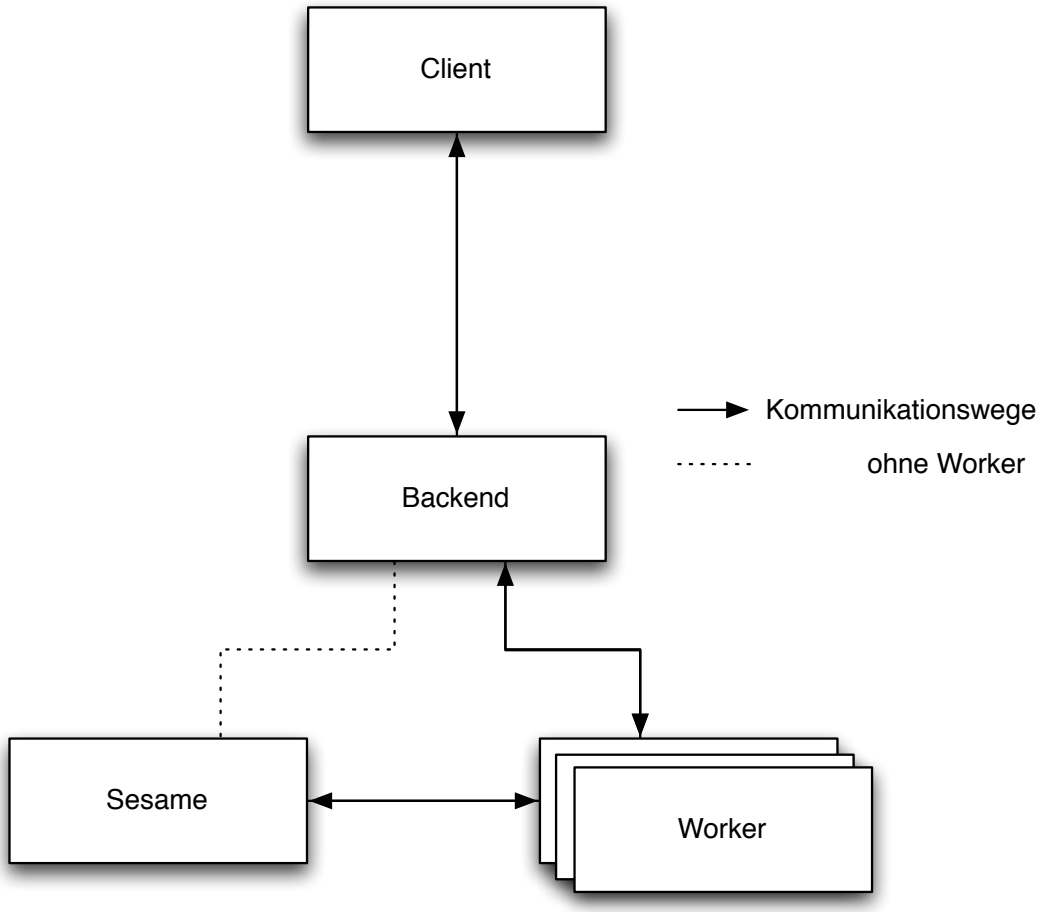

Abbildung 8: Struktur der Anwendung

6.5.3 Das Serverbackend

Das Backend verwaltet die Datenbestände der einzelnen Pipes, der Nodes pro Pipe, der Kantenverbindungen (Edges), der Nutzerdaten und der eingebunden RDFGraphen pro Nutzer

Die Beantwortung der Anfragen an die vorgeschaltete Informationsseite wird durch den SiteController

```
(app/controllers/site_controller.rb)
```

übernommen.
Für die Operationen der einzelnen Programmteile stehen derweil eigene Controller bereit, die für den eingeloggten Nutzer auf seinen Daten CRUD[17] Operationen nach einen REST-basierten Interface bereitstellen.

[17]CRUD: Create, Read, Update and Delete, http://en.wikipedia.org/wiki/Create,read,updateanddelete

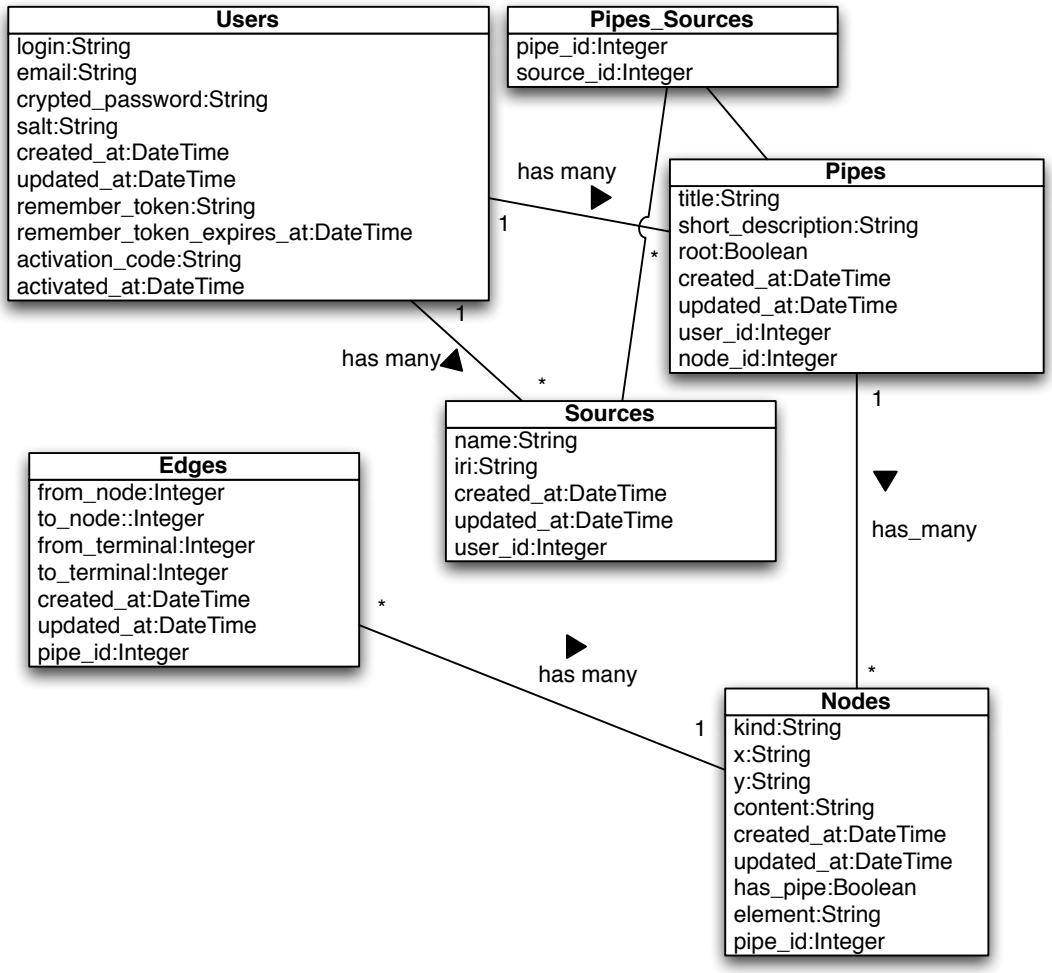

Abbildung 9: Struktur der Anwendung (Datenbank)

6.5.4 Sesame und die Anbindung an Graphpipes

Networked Graphs sind in der aktuellen Version als Sail (NGSail[18]) in Sesame implementiert. Sesame ist ein RDF basiertes Reporsitory und Anfragewerkzeug. Die in Graphpipes gestalteten Pipes werden in SPARQL konvertiert und an Sesame übergeben. Sollte eine Pipe weitere RDF-Daten benötigen, so werden diese zuvor in Sesame hochgeladen und gespeichert. Die aus der Pipe resultierenden Triples oder – im Falle eines *Networked Graphs* – RDF-graphen werden an Graphpipes zurückgeliefert und an den Browser übermittelt

6.5.5 Strukturbeschreibung der Anwendung auf Backendseite

Rails basiert auf dem Model-View-Controller Pattern. Die Persistenz der Daten wird über die Modelklassen (/app/models) durch die Speicherung in der Datenbank gewährleistet. Zusätzlich wird in den Model-Objekten auch die Verbindungen zwischen den einzelnen Modellen definiert.

```
(app/controllers/site_controller.rb)
class Pipe < ActiveRecord::Base
  belongs_to :user
  has_many :nodes, :dependent => :destroy
  has_many :edges, :dependent => :destroy
  belongs_to :node
  has_and_belongs_to_many :sources
end
```

Teil des Sourcecode aus app/models/pipe.rb (vgl. Fig. 9)

Die Controller nehmen die Aufgabe der Koordination zwischen der Anfrage, der Sammlung der Daten aus den Modellen und der Auslieferung in den Views wahr. Die Views werden im letzten Schritt der Anfrage an den Browser geschickt, der diese dann, im Falle von HTML, anzeigt, beziehungsweise im Falle von JSON als Javascriptanweisung interpretiert.

6.5.6 Models

Graphpipes besitzt sieben Model-Klassen: Zunächst werden die mit Edges, Nodes, Sources und Pipes (pipes.rb, nodes.rb, sources.rb, pipes.rb) Bestandteile eines Graphen definiert. Der exakte Aufbau ist in Abbildung 9 dargestellt:

Ein User kann mehrere (0..n) Pipes besitzen. Diese bestehen aus Nodes, Edges und Sources. Ob der Nutzer eingeloggt ist und welche Rechte er besitzt wird in der Sessionklasse gespeichert. Für jede in der Anwendung erzeugte Node und Kante und für jeden Verweis auf einen RDF-Graphen wird ein Eintrag in der Datenbank erzeugt. Die weiteren zwei Klassen UserMailer und UserObserver dienen der Kommunikation mit dem User,

[18] NGSail von Simon Schenk, https://launchpad.net/networkedgraphs

sollte er denn beispielsweise sein Passwort vergessen haben und dieses neu anfordern müssen.

Graphpipes in seinem aktuellen Entwicklungsstand greift auf SQLite3[19] als Datenbank zurück. Sollte es im Laufe der Zeit notwendig sein, auf eine leistungsfähigere Datenbank umzusteigen, so kann dies ohne Probleme durchgeführt werden.

Wird nach der Installation die Datenbank initialisiert, werden automatisch Testdaten migriert. Graphpipes stellt drei Beispielpipes, sowie einen Testnutzer (admin/secret) zur Verfügung.

6.5.7 Controller

Alle Controller innerhalb einer Railsanwendung erben von ActionController::Base. Diese Klasse stellt die Grundlagen für die Verarbeitung des Requests, die Konvertierung der Requestparameter (Beispielsweise /test=hallo%20welt in params[:test] mit dem Wert 'Hallo Welt') und die Weitergaben der Instanzvariablen an die ViewObjekte weiter. Im Allgemeinen ist es üblich, eine ApplicationController Klasse einzufügen, von der dann ihrerseits alle weiteren Controller erben. So kann man für alle Controller gültige Funktionen zentral an einer Stelle implementieren. Graphpipes nutzt dies, um die Authentifizierungscode in allen Controller zur Verfügung zu stellen.

6.5.8 Resourcen

Alle Operationen an Pipes werden über REST Transformationen in Form einer einfachen API gewährleistet: Dies sind PipesController, NodesController und EdgesController. Die Controller stellen ihre Funktionalität in Abhänigkeit voneinander zur Verfügung und bilden einen Resourcenbaum mit folgendem Aufbau:

```
User > Pipe > Nodes | Edges
```

Um die Resourcen ansprechen zu können, muss der Nutzer eingeloggt sein und über die entsprechenden Rechte besitzen, d.h. er muss der Ersteller der Pipe sein.

Die API wird durch das Clientfrontend direkt genutzt, kann aber auch auf anderem Weg von weiteren Anwendungen genutzt werden. Hier einige Beispiele:

Eine XML-formatierte Liste aller Nodes der Nutzers Johann mit der Nutzer ID 12 kann man auf folgen Art erhalten.

```
curl -X GET --basic johann:passwort http://graphpipes.de/users/12-johann/pipes.xml
```

Eine neue Node kann man zu einer Pipe (beispielsweise die Pipe mit der ID 22) hinzufügen:

```
curl -X POST -d 'node[x]=50; node[y]=200;node[kind]=repo;node[has_pipe]=false' \
--basic johann:passwort http://graphpipes.de/users/12-johann/pipes/22/nodes
```

Das Löschen einer Node mit ID 44 geschieht so:

```
curl -X DELETE --basic johann:passwort http://graphpipes.de/users/12-johann/pipes/22/nodes/44
```

[19] SQLite3 ist eine serverlose SQL Datenbankengine. http://www.sqlite.org/

6.5.9 PipesController::show - Die Arbeitsfläche

Zentraler Angelpunkt der Arbeitsflächenansicht ist die Methode *show* des PipesControllers.

```
def show
  @parent_pipes = find_parental_pipe
  @pipe = current_user.pipes.find_by_id params[:id]

  raise ActiveRecord::RecordNotFound if @pipe.nil?

  render :action => 'show', :layout => 'stage'
end

def find_parental_pipe
  return nil if params[:parent].nil?
  current_user.pipes.find_all_by_id(params[:parent])
end
```

Jede Pipe kann für sich alleine stehen, Bestandteil einer weiteren Pipe sein oder aber selbst weitere Pipes als Subelemente besitzen. Zuallererst wird deshalb geprüft, ob eine übergeordnete Pipe vorhanden ist und diese gegebenfalls auch aus der Datenbank angefordert. Als nächstes wird die gewünschte Pipe geladen und die View (/app/views/pipes/show.html.erb) samt des Stagelayouts (/app/views/layouts/stage.html.erb) gerendert. Das Layout übernimmt die allgemeine Gestaltung der Arbeitsfläche: Die CSS-Buttonslayouts, die Werkzeugpalette, die Javascriptsdateien für die Funktionalität der Anwendung werden geladen. Innerhalb des show.html.erb Templates werden die Daten für die darzustellenden Pipe gerendert:

```
<script>
window.boxes = {
    containers: [
      <% @pipe.nodes.each_with_index do |node, index| \%>
        <% has_pipe = node.has_pipe.blank? ? false : "'#{node.has_pipe.id}'" %>

        <%= "{xtype: #{node.kind}, id: #{node.id}, position: [#{node.x.to_i}, #{node.y.to_i}],
        codeText: '#{node.content}' + ' ' + #{node.id}, element: '#{node.element}',
        node_id: #{node.id}, pipe_id: #{node.pipe.id}, user_id: #{node.pipe.user.id},
        has_pipe: #{has_pipe} }"%><% if index < @pipe.nodes.size - 1 \%>,<\% end %>
      <% end %>
    ],

    wires: [
    <% @pipe.edges.each_with_index do |edge, index| %>
      <%= "{id: #{edge.id}, src: {moduleId: #{edge.from_node}, terminalId: #{edge.from_terminal}},
      tgt: {moduleId: #{edge.to_node}, terminalId: #{edge.to_terminal}}}" %>
      <% if index < @pipe.edges.size - 1 %>,<% end %>
    <% end %>
    ]

 }
</script>
```

Auf diese Weise wird die gesamte Datenstruktur der Nodes und ihrern Verbindungslinien an den Javascriptinterpreter auf der Clientseite übergeben. Die Konstruktion der

Elemente auf der Arbeitsfläche erfolgt innerhalb der, als letztes innerhalb des stagelayouts eingebundenen, stage.js. Zugleich werden an dieser Stelle, wie bereits in Kapitel 6.5.1 beschrieben, die Nutzer- und PipeID dynamisch in das Javascriptfile eingerechnet.

```
<script src="/javascripts/stage/<%= current_user.id %>/<%= @pipe.id %>.js"
type="text/javascript" charset="utf-8"></script>
```

6.5.10 PipeController::run

Soll die Pipe ausgeführt werden, wird die SPARQL-Query vom Client per HTTP-POST-Request an das Backend geschickt. Die entsprechende Methode für die weitere Verarbeitung dieser Anfrage ist im PipesController::run zu finden. Um den restbasierten PipesController um einen POST-Request auf http://graphpipes.de/user/12/pipes/12/run zu erweitern müssen zuvor die Routes in config/routes.rb angepasst werden. *Run* wird als Operation auf einen Element der Resourcesammlung Pipes definiert:

```
user.resources :pipes, :member => {:run => :post}
(aus config/routes.rb)
```

Die Anfrage wird an Sesame mit der Bitte, als Antwort JSON (Javascript Objekt Notation) zu erhalten (per ACCEPT-HEADER definiert) übermittelt. Sobald die Query bearbeitet von Sesame zurückgeliefert wird, wird sie zusammen mit einem entsprechendem Statuscode an das Frontend weitergeleitet, welches Aufgrund des JSON-Formates ohne Formatkonvertierung in der Lage ist, diese weiterzuverarbeiten.

Um alle Anfragen und Ergebnissgraphen zu speichern und den Queries eindeutig wieder zuzuordnen, werden diese in /public/responses als Dateien gespeichert. Als Dateiname kommt der SHA1-Hash über die Query zur Verwendung. So kann jede gleiche Query immer der gleichen Datei zugeordnet werden. Wie später noch beschrieben, bestünde an dieser Stelle die Möglichkeit, die Queries versioniert abzuspeichern, um den Entwicklungsverlauf und die Veränderungen der einzelnen Queries analysieren zu können. Ein weiterer Optimierungspunkt ist die Auslagerung des langlaufenden, blockenden IO-Prozesses an einen vom Railsprozess unabhängigen Workerthread. Da Rails nicht bzw. nicht vollständig threadsafe ist (vgl. [Rub06]) und Ruby desweiteren keine native Betriebssystemthreads unterstützt (vgl. [Sch07]), sondern nur eine vereinfachte Threadingengine (Green Threads) ref innerhalb des Prozesses unterstützt, kann jeder Railsprozess immer nur eine Anfrage gleichzeitig beantwortet. Läuft die Einfrage an Sesame nun länger, ist auch das Backend nicht erreichbar. Eine Lösungsvariante auf Basis von Twitters Starling[20] wurde bereits in Graphpipes integriert

[20] http://twitter.com/, Starling: http://rubyforge.org/projects/starling/

6.5.11 Das Frontend

Das Frontend wird durch zwei Controller innerhalb der Railsapplikation gesteuert. Für die Gestaltung und das Layout zeichnet sich der SiteController

`(app/controllers/site_controller.rb)`

verantwortlich. Er stellt die Informationsseiten und das Impressum dar und leitet eingeloggte Nutzer auf ihre Dashboard – die Übersicht ihrer Pipes weiter.

Die Logik der clientseitigen Anwendung wird über den zweiten Frontendcontroller gesteuert:

`javascripts_controller.rb`

. Er stellt auf Anfrage die Javascriptfiles bereit.

6.5.12 Javascripts

Es stehen für die meisten Interfaceelemente Javascriptojekte zur Verfügung. Zu finden sind die auf Basis des Frameworks prototype.js[21] geschrieben Klassen[22] in app/views/javascripts/*.

So wird die Gallery auf der Startseite von gallery.js.erb gesteuert, die Erstellung der Buttons über button.js.erb. Desweiteren gibt es für die im parallelen Interaktionsmodel (siehe Kapitel 6.3) verwendeten Dialogboxen bzw. Sheets einen eigene Klasse. Die Darstellung der Nodes und ihrer Verbindungskante wird über Wireit.js.erb bzw, der Kindklasse NodeBox.js.erb, einem Framework basierend auf Yahoo! Pipes[23] gesteuert.

Diese beiden Klassen sind im Laufe der Anwendung für die Kommunikation mit und der Persistierung ihres States auf der Backendseite verantwortlich: Wird eine Kante oder eine Node verschoben, gelöscht oder neu erzeugt, so wird noch vor der Darstellung auf der Arbeitsfläche die Änderung serverseitig abgespeichert. Im Fehlerfalle wird keine Veränderung der Daten sowohl im Front- als auch im Backend vorgenommen. Auf diese Weise wird dafür garantiert, das die beiden Seiten datensynchron sind.

Neben den GUI-Klassen gibt es zwei zentrale Objekte, die den Kontrollfluß der Clientanwendung maßgeblich bestimmen: parser.js.erb und stage.js.erb.
parser.js.erb ist für den Interpretation und die Erstellung der SPARQL-Query aus dem Nodegraphen verantwortlich. Die Klasse stellt desweiteren Funktionen zur Verfügung um zu erkennen, ob es sich um einen Networked Graph handelt.
Zentraler Bestandteil der Analyse ist der rekursive Durchlauf der in Javascript gebildeten Objektbaumstruktur (Abb. 10):
Jede Node besitzt eine results()-Methode, die ihren State in SPARQL serialisiert. Der Baum wird nun von der Wurzel aus abgelaufen. Folgt als nächstes eine CONSTRUCT-Node, so ist der Objektgraph als Networked Graph zu interpretieren. Im weiteren Verlauf

[21] Das prototype javascript framework von Sam Stephenson: http://www.prototypejs.org/

[22] Anmerkung: Da Javascript einen prototypbasierte Sprache ist, gibt es im eigentlichen Sinne keine Klassen, sondern nur Objekte

[23] http://pipes.yahoo.com/pipes/

wird nun jede Node getestet, ob sie weitere Kindknoten besitzt und diese als eigenständiger Baum rekurisv analysiert. Die gesammelten Daten werden als Sammlung von NodeID und Node-serialisierung zurückgeliefert.

```
var Parser = Class.create({
[..]
 parse : function(a_node) {
     if (!this.tree_exists(a_node)) { return [{id:0,result:0}] }

     var node    = a_node
     var results = []
     var me      = this

     node.find_input_terminals().each(function(terminal) {
       terminal.wires.each(function(wire) {

         other_node = wire.getOtherTerminal(terminal).nodeBox
         results.push({id: other_node.config.node_id, result: other_node.result()})

         if (me.has_children(other_node)) {
           var tree_result = me.parse(other_node)
           results = results.concat(tree_result)
         }
       })
     })
     return results
}
[..]
}
aus app/views/javascripts/parser.js.erb
```

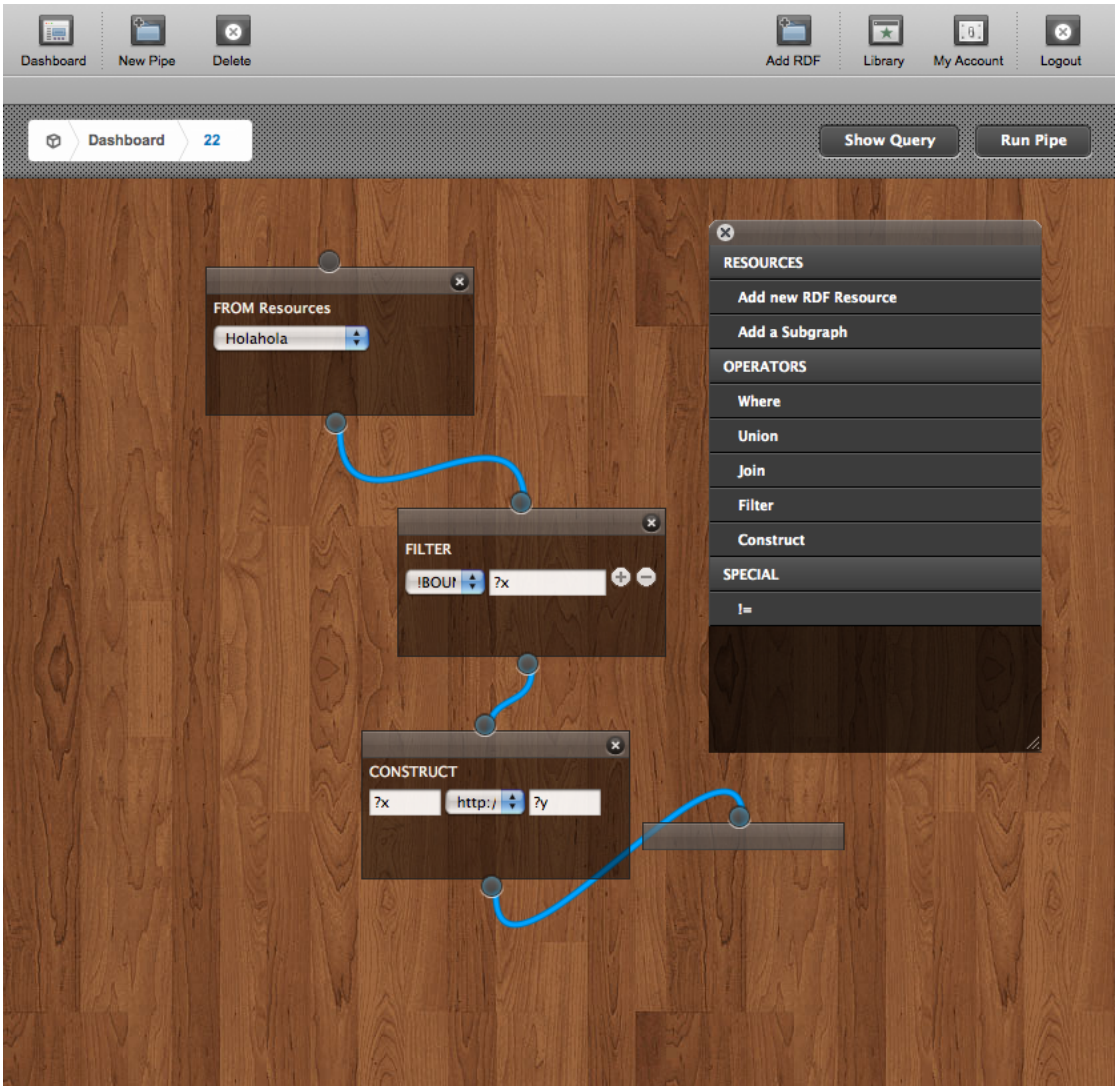

Abbildung 10: Beispielgraph auf der Arbeitsfläche

6.5.13 Exkurs: Kommunikation via REST-API

REST (Representational State Transfer) ist eine von Roy Fielding[24] in seiner Dissertation vorgestellte Sammlung von Prinzipen, die ein allgemeines Vorgehen, Änderungen an Datensaetzen sowie die Abfrage von Daten, beschreiben. Die Datenbestände werden als Resourcen bezeichnet und REST-basierte Systeme stellen auf Basis der HTTP-Verben (DELETE, PUT, POST und GET) die entsprechenden Operationen auf diesen Daten zur Verfügung.

Sowohl die Kommunikation zwischen Client und Backend, als auch zwischen Backend und Sesame wird per REST durchgeführt. Die in Rails eingesetzte Interpretation von REST ermöglicht es, dass Resource automatisch gefiltert und gescoped werden. Die URLPfade zu der Kante einer Pipe eines Nutzers geschieht in Folgender Form.

```
/users/1/pipes/2/edges/3
```

Um eine Kante zu erreichen, muss sowohl der Nutzer als auch die Pipe angegeben werden. Es ist nicht möglich per /edges/3 direkt auf die Kante zuzugreifen. Dadurch in Zusammenspiel mit einer Sessionbasierten Authentifikation kann auch eine einfache umzusetzenden Zugriffsbeschränkung umgesetzt werden.

```
class EdgesController
  [..]
  def show
    @pipe = current_user.pipes.find_by_id(params[:pipe_id])
    @edge = @pipe.edges.find_by_id params[:id]

    raise ActiveRecord::RecordNotFound if @edge.nil?
   [..]
  end
end
```

currentuser ist der aktuell eingeloggte User. Dieser ist an dieser Stelle im Programmlauf immer gesetzt, denn ohne gültige Zugangsdaten kann die Aktion show nicht ausgeführt werden. Gewährleistet wird das durch einen zuvor ausgeführten Filter:

```
before_filter :login_required
```

Die Berechtigungsüberprüfung wird durch das Railsplugin *restful authentication* [25] bereitgestellt.

Jeder User hat folgende Verknüpfungen zu anderen Tabellen/Modellen (vgl. Abbildung 9):

```
class User
  has_many :pipes, :dependent => :destroy
  has_many :nodes, :through => :pipes
  has_many :edges, :through => :pipes

  has_many :sources
end
```

[24] Roy Fielding: http://www.ics.uci.edu/ fielding/

[25] http://github.com/technoweenie/restful-authentication/tree/master

Jeder user hat 0..n Pipes, die bei Löschung des Nutzers mitgelöscht werden (definiert durch :dependant :destroy), sowie 0..m nodes und 0..b edges, die über das Pipes Model geproxied werden.

Zuallerest wird in der Showaktion die zur id passende Pipe, dann die gewünschte Kante gesucht. Sollte die Kante nicht existieren, wird dem Anwender eine HTTP 404 Fehlermeldung angezeigt.

7 Schlussbetrachtung

Kann Graphpipes als graphisches Werkzeug, um komplexe Queries und Networked Graphs zu bilden funktionieren? Wie bei vielen Versuchen, abstrakte Konzepte aus einem textbasierten, direkten Kontext in einen graphischen Zusammenhang zu bringen, ist diese Frage nicht leicht zu beantworten, denn es kommt auf die Aufgabenstellung und die Nutzerzielgruppe an.

Man muss natürlich im Vergleich zur Eingabe und der händischen Konstruktion von Queries Kompromisse eingehen, denn ab eines gewissen Grades an Komplexität, wächst der Umfang und der Anspruch an die graphische Lösung an. Schnell machen sich Schwächen im Abbildungskonzept bemerkbar und man stößt an die Grenzen des Machbaren. Für einfache Konstrukte und um auf spielerische Art und Weise das Prinzip hinter Networked Graphs und SPARQL zu erklären ist eine graphische Herangehensweise sicherlich einfacher zu vermitteln.

Ein weiteres Problem tritt in der Frage zu Tage, ob es überhaupt sinnvoll und praktikabel ist, Anwendungen im Desktopstil in den Browser zu verlagern? Neben der Übernahmen eines Metaphern- und Funktionskonzeptes verlangt eine solche Applikation einen höheren technischen Aufwand als ein vergleichbares lokal funktionierendes Interface: In der zustandslosen Welt des Internets ist es der Nutzer nicht gewohnt einen langandauerenden Prozess innerhalb einer einzelnen Seite zu vollziehen. Hinzu kommt das Problem, dass viele Interfaceelemente wie Fenster, im Falle von Graphpipes die Werkzeugpalette, zwar ihrem Pendant im Betriebssystem entsprechen, sich aber anders verhalten. Die Werkzeugleiste kann nicht auf einen zweiten Monitor gezogen oder über anderen Fenster gelegt werden. Der Anwendung und der Nutzer sind fest an den Browser gebunden.

7.1 Ausblick und Fazit

Graphpipes sollte in sofern umgestaltet werden, dass die Analyse der Graphen nicht mehr auf Browserseite von statten geht. Zum einen könnten so Graphen ohne an einen Browser gebunden zu sein interpretiert und ausgeführt werden und es ist auf diese Weise möglich auch andere Frontends zu implementieren, ohne jedesmal den Parser neu entwerfen zu müssen, es würde auch die rekursive Analyse von per Networked Graphs eingebundenen Graphen erheblich erleichert.

In einem weiteren Entwicklungsschritt wäre es zudem wünschenswert, weitere SPARQL-Elemente zu implementieren. Zudem könnte überprüft werden, ob es sinnvoll ist, das Interface noch stärker zu vereinfachen und auf den direkten Bezug zu SPARQL und Networked Graphs zu verzichten und diese eher als Mittel zur Durchführung der Analyse und nicht als Interfacebildend anzusehen.

Zuletzt besteht die Möglichkeit, die gespeicherten Pipes, Graphen und Queries regelmäßig oder sollte sich ein Datenquelle geändert haben, auszuführen und die Ergebnisse versioniert mit einer Zugriffsmöglichkeit über einen Permalink anzubieten, damit nicht nur die Aktualität gewährleistet ist, sondern diese auch als Resource genutzt werden kann, die sich nicht verändert. Änderungen könnten per RSS-Feed verbreitet werden. Dies würde zusammengenommen die Möglichkeit eröffnen, nicht nur einen Schnappschuss

der Daten zu analysieren, sondern auf die zeitlichen Beziehungen und Veränderungen zu interpretieren.

7.2 Fehlende Implementation

Einige Bestandteile von Graphpipe konnten noch nicht implementiert werden. Hierzu gehören wie folgt:

- Der Upload von RDF Daten in das Sesame Repository. Der Mechanismus für den Transfer per REST-Interface ist vorhanden, allerdings wird die Pipe noch nicht auf die entsprechenden Resourcenodes überprüft.
- Edges (Kanten) werden nicht gezeichnet und teilweise auch noch nicht richtig an das Backend übermittelt.
- Die Pipe wird bisher ananlysiert, die richtige Anordnung der dadurch erzeugten SPARQL-Bestandteile wird nicht vorgenommen.
- Legt man einen Subgraph an, wird noch keine entsprechende neue Pipe angelegt
- Subgraphen werden nicht rekursiv analysiert. Dafür müsste entweder der entsprechende Subgraph dynanisch per iFrame geladen und analysiert werden oder die Analyse der Graphen serverseitig von statten gehen
- Im Laufe der Ausarbeitung ergab sich zudem ein weiteres Problem, dass am 19.06.2008 die Verbreitung der Zuvor frei verfügbaren WireIt Bibliothek (http://javascript.neyric.com/wireit/) auf Druck von Yahoo! eingestellt wurde. Der Author verweisst auf die unklare Rechtslage[26].

[26] Nachzulesen unter http://javascript.neyric.com/blog/2008/06/19/wireit-removed-from-distribution/

8 Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

Koblenz, den 04.08.2008

Thomas Winkler

9 Literatur

Literatur

[Abo] Abobe. Flash player penetration. *http://www.adobe.com/products/playercensus/flashplayer/*.

[BL98] Tim Berners-Lee. Semantic web road map. 1998.

[BL99] Tim; Mark Fischetti Berners-Lee. *Weaving the Web: Origins and Future of the World Wide Web*. Number ISBN 0-7528-2090-7. Orion Business, 1999.

[Fow02] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[htt] http://www.w3.org/2004/03/trix/intro. W3c: Named graphs.

[htt08a] http://pipes.yahoo.com/pipes/. Yahoo! pipes, 2008.

[htt08b] http://rubyonrails.org/. Ruby on rails, 2008.

[JN08] Donald A. Norman Jakob Nielsen. Usability on the web isn't a luxury, 2008.

[ML04] Jason Fried Matthew Lindermann. *Defensive Design for the Web*. New Riders, 2004.

[NS06] Tim Berners-Lee Nigel Shadbolt, Wendy Hall. The semantic web revisited. *T he Semant i c Web*, 2006.

[Ope] OpenRDF. Sesame, http://www.openrdf.org/.

[rdf] *http://www.w3.org/TR/rdf-sparql-query/*. W3C.

[Rub06] RubyForum. Making rails thread safe, priority or not? 2006.

[Sch07] Werner Schuster. The futures of ruby threading. *http://www.infoq.com/news/2007/05/ruby-threading-futures*, 2007.

[Sch08] Simon Schenk. Networked graphs: A declarative mechanism for sparql rules, networked graphs: A declarative mechanism for sparql rules, sparql views and rdf data integration on the web. *WWW 2008 / Refereed Track: Semantic / Data Web - Semantic Web II*, 2008.

[Usc01] Michael Uschold. Where are the semantics in the semantic web? *http://www.starlab.vub.ac.be/WhereAreSemantics-AI-Mag-FinalSubmittedVersion2.pdf*, 2001.

[W3C08] W3C. Sparql query. 2008.

[Wil03] Russell B. Williams. Web surfing ? how about a new analogy for experiencing the web?part, 08 2003.

10 Anhang

10.1 Installation

About graphpipes graphpipes is a simple and easy way to aggregate semantic data.
By using drag'n'dropin a web-based application you are able to create SPARQL-queries and reuse them later in your networked graphs.
graphpipes is a proof of concept, developed at University of Koblenz-Landau, Germany
Requirements

1. Ruby 1.8.2+
2. Rails 2.0.2+
3. Linux
4. A well written Browser (Firefox, Safari, Opera)

Quick Install

1. Download graphpipes (master branch on github: http://github.com/tomfarm/graphpipes/tree/master)
2. cd into the graphpipes directory
3. Rename config/database.sample.yml to database.yml (you may want to change to config to your needs)
4. rake db:migrate
5. ./script/server -e production
6. visit http://localhost:3000
7. You may login using 'admin/secret'

Thomas Winkler - Koblenz, 03/30/2008 tomfarm@gmail.com, twinkler@uni-koblenz.de

10.2 MIT-Lizenz

http://www.opensource.org/licenses/mit-license.php

The MIT License

Copyright© 2007-2008 Thomas Winkler Permission is hereby granted, free of charge, to any person obtaining a copyof this software and associated documentation files (the "Software"), to dealin the Software without restriction, including without limitation the rightsto use, copy, modify, merge, publish, distribute, sublicense, and/or sellcopies of the Software, and to permit persons to whom the Software isfurnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included inall copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS ORIMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THEAUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHERLIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS INTHE SOFTWARE.

10.3 Screenshots

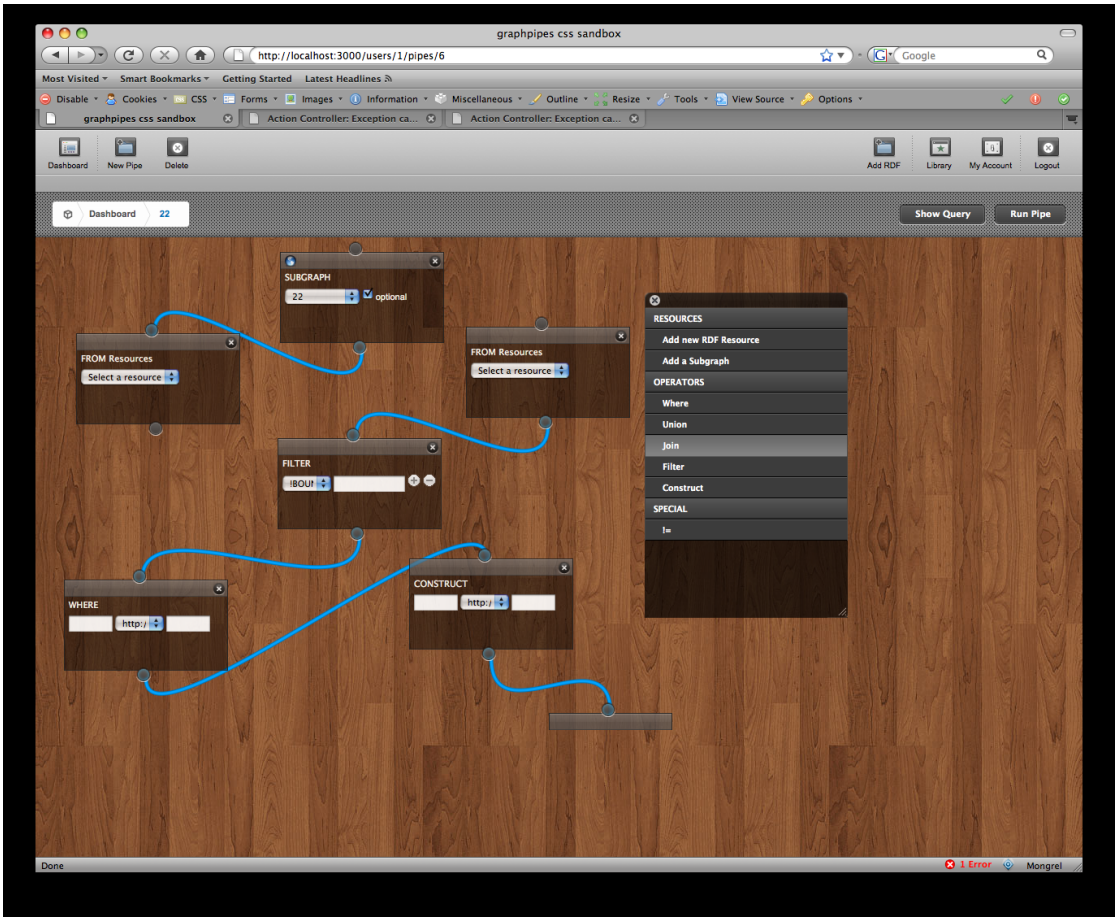

Abbildung 11: Beispielgraph auf der Arbeitsfläche

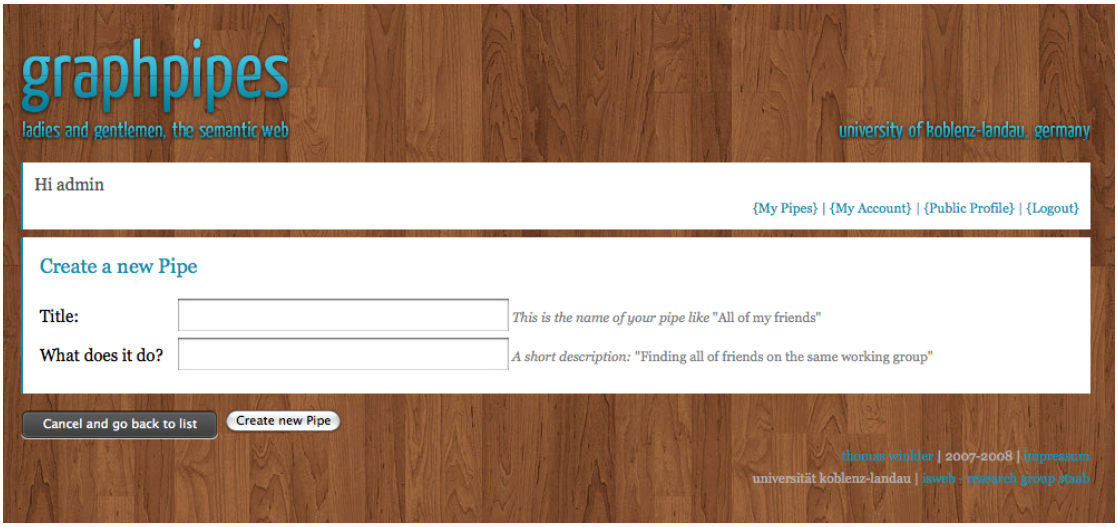

Abbildung 12: Das Dashboard mit einer angelegten Pipe

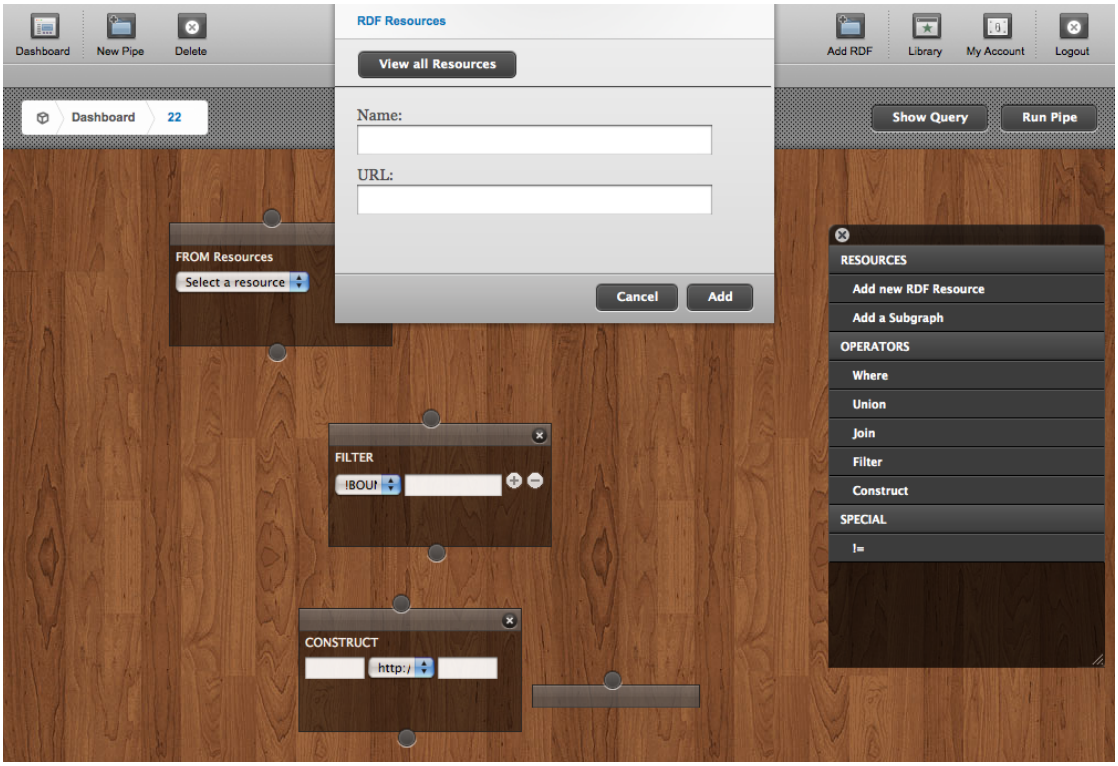

Abbildung 13: Arbeitsfläche mit Sheets zur Eingabe von Verweisen zu RDF-Graphen

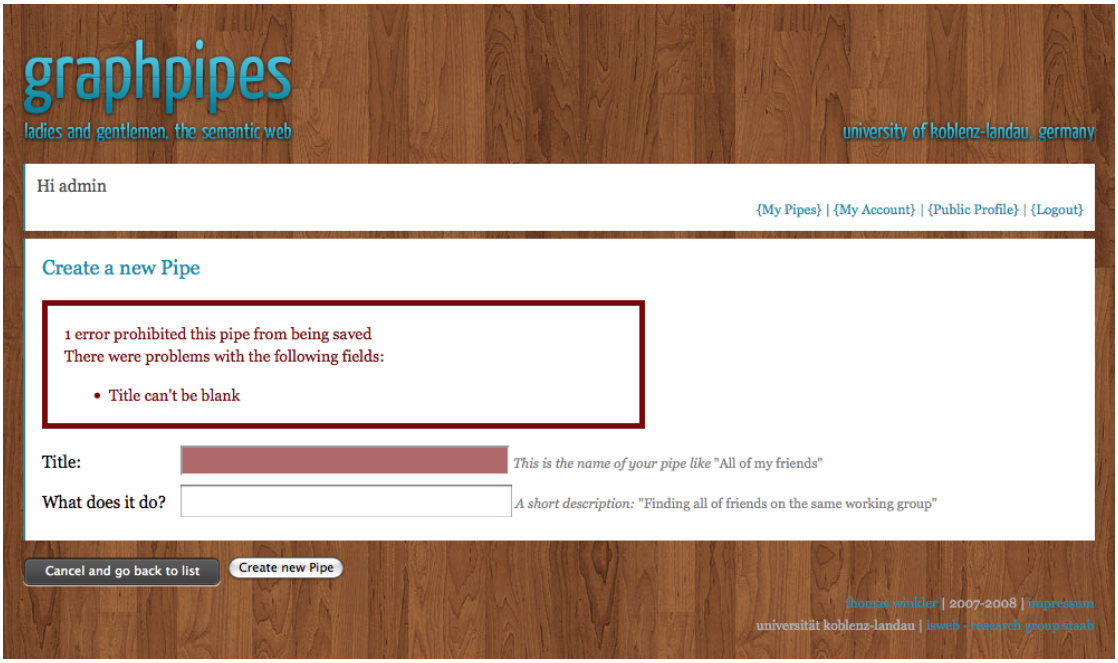

Abbildung 14: Erstellung einer neuen Pipe mit Fehleranzeige

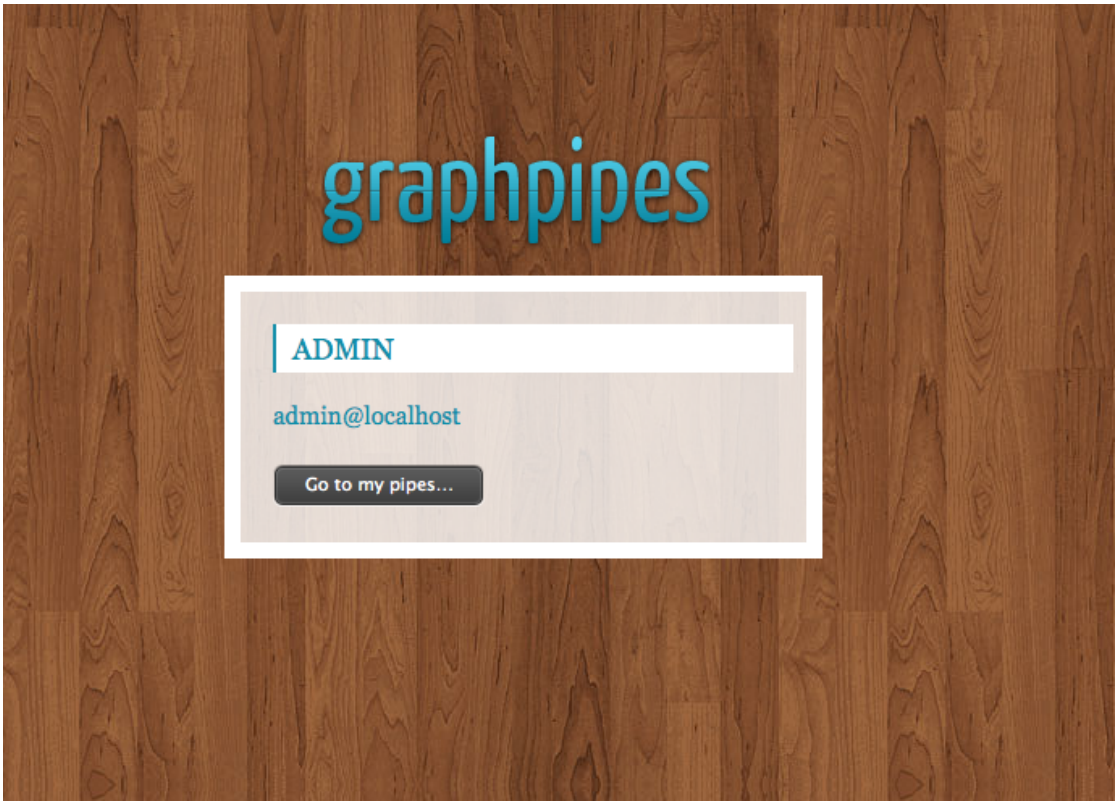

Abbildung 15: Informationsscreen des Users (öffentlich)

Abbildung 16: Informationsscreen des Users (nicht öffentlich)

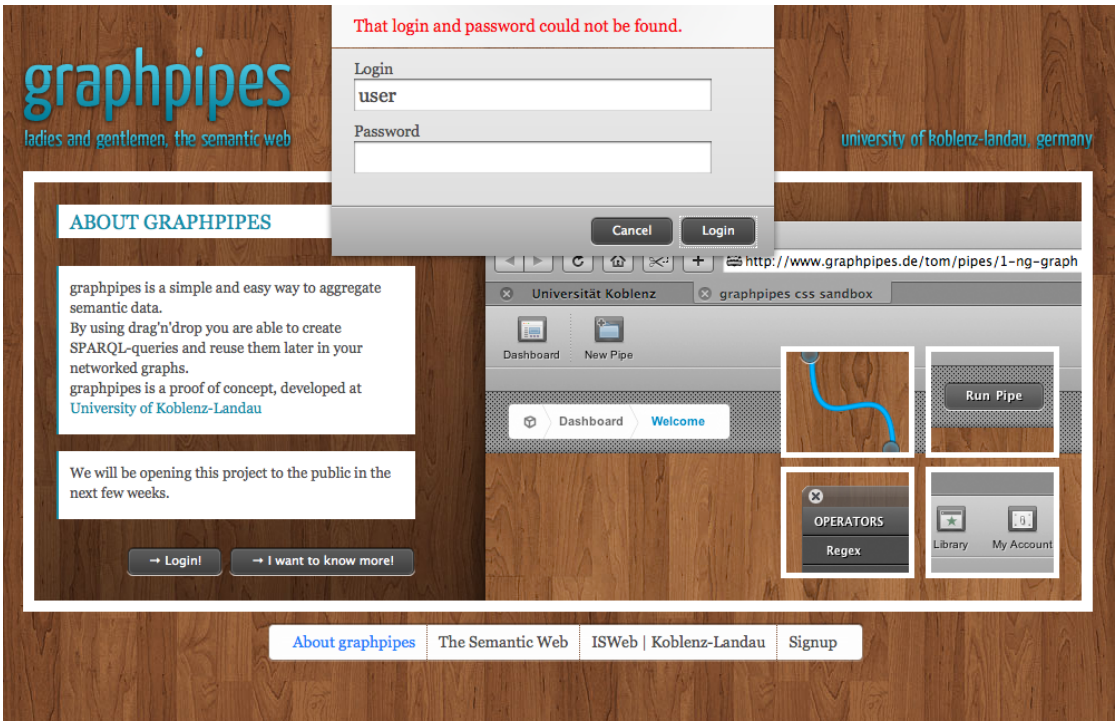

Abbildung 17: Login auf der Willkommensseite mit Fehleranzeige

Abbildung 18: Willkommensseite Teil 1

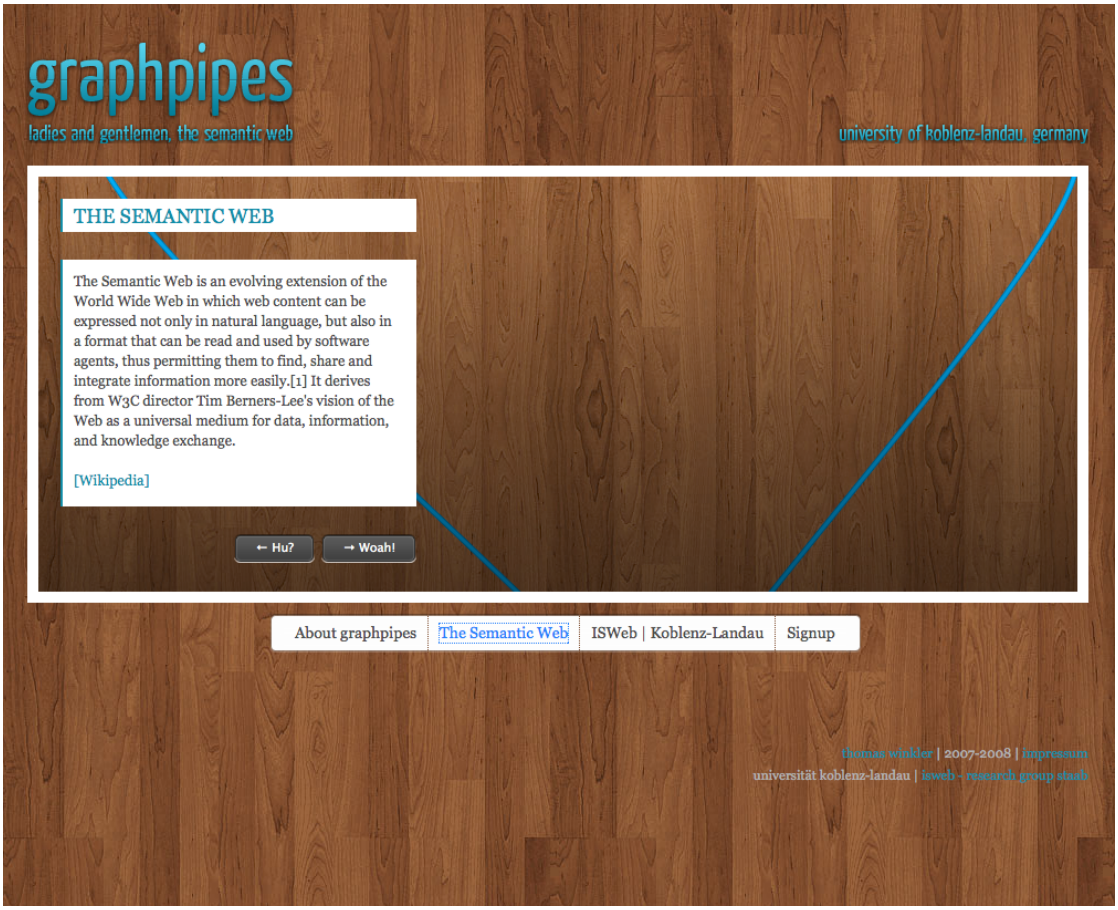

Abbildung 19: Willkommensseite Teil 2

Abbildung 20: Willkommensseite Teil 3

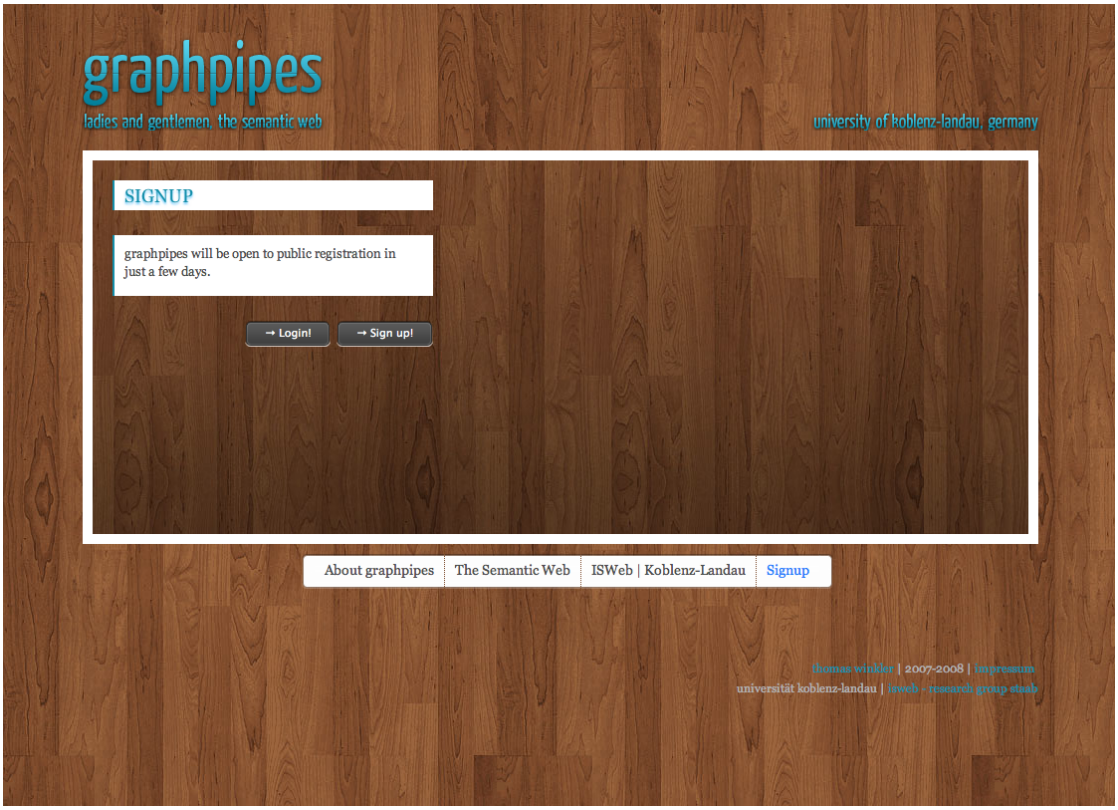

Abbildung 21: Willkommensseite Teil 4