Aufgabenblatt 5 - Entwurfsbeschreibung des Fremdprojektes

Inhaltsverzeichnis

1 iSPARQL

1.1 Allgemeines

iSPARQL ist eine Webanwendung zur visuellen Erstellung von SPARQL-Anfragen. Die Bedienung erfolgt als JavaScript-unterstützte Website im Browser, wobei die Anwendung auf Datenressourcen über die SPARQL-Schnittstelle zugreift.

1.2 Produktübersicht

Die Erstellung von gültigen SPARQL-Anfragen erfordert einiges an Fachwissen und birgt auf Grund der kryptischen Notation allerlei Fallstricke und Anfälligkeiten für Tippfehler. Beide Problem versucht iSPARQL mittels eines bildlichen Editors für Anfragen zu vermindern. Zudem soll so ein einfacherer Einstieg und eine intuitivere Herangehensweise zum Umgang mit SPARQL-Ressourcen geschaffen werden.

An zentraler Stelle steht beim Entwurf der Programmoberfläche deshalb ein Zeichenbereich, in dem Ressourcen und Restriktionen in Form graphischer Objekte repräsentiert werden. URIs oder Literale werden durch Kreise dargestellt, Einschränkungen mittels Connector"genannter Verbindungen zwischen diesen. Die flüssige Bedienung erfolgt dabei mittels Maus über eine gefällige, strukturierte Oberfläche mit Menüelementen und einer Werkzeugleiste und vermittelt den Komfort eine Desktop-Anwendung.

1.3 Grundsätzliche Struktur und Entwurfsprinzipien für das Gesamtsystem

Entwurfsprinzipien von iSPARQL

iSPARQL ist ein in JavaScript realisierter visueller, SVG-basierter SPARQL-Query-Generator und gleichzeitig auch Ergebnis-Browser. Es ist Teil des OpenLink AJAX Toolkit (OAT) und verwendet dessen Funktionen zur Darstellung von UI-Elementen wie Widgets und Kontrollelementen, wie auch zum Event Management und zur Anbindung einer Datenquelle, in diesem Fall SPARQL, mittels der OAT AJAX Database Connectivity.

Die Anwendung besteht aus der eigentlichen Website mit Layoutangaben im HTML-Format, den OAT-Klassenbibliotheken und den iSPARQL-spezifischen Scripten. Ein klares Entwurfsprinzip ist dabei nicht erkennbar, es enthält jedoch Elemente des Model-View-Controller-Modells.

Das Open Link AJAX-Toolkit

Das Open Link AJAX-Toolkit bildet die Grundlage für das grafikbasierte Anfrage-Tool i(nteracive)SPARQL. Es stellt verschiedene Oberflächenelemente für das User Interface bereit. Hierzu zählen Combo-Boxen, einfache Texteingabe Felder (Quickedit) oder etwa Date Picker. Der zur Verfügung gestellte WebDAV Browser ermöglicht es dem Benutzer durch die WebDAV Instanzen zu navigieren als befände er sich in einem lokalen Dateisystem. Außerdem sind Methoden implementiert, die es ermöglichen RDF-Daten als Graph zu visualisieren. Dies wird in iSPARQL zur Darstellung der Anfrage ausgenutzt.

Die AJAX Database Connectivity stellt für das AJAX-Toolkit eine Datenbank unabhängige Datenzugriffsschicht zur Verfügung. Hierdurch wird u.a. iSPARQL in die Lage versetzt

komplexe SPARQL Anfragen zu generieren, da sie sich nicht mit der Komplexität des Datenzugriffs auseinandersetzen muss.

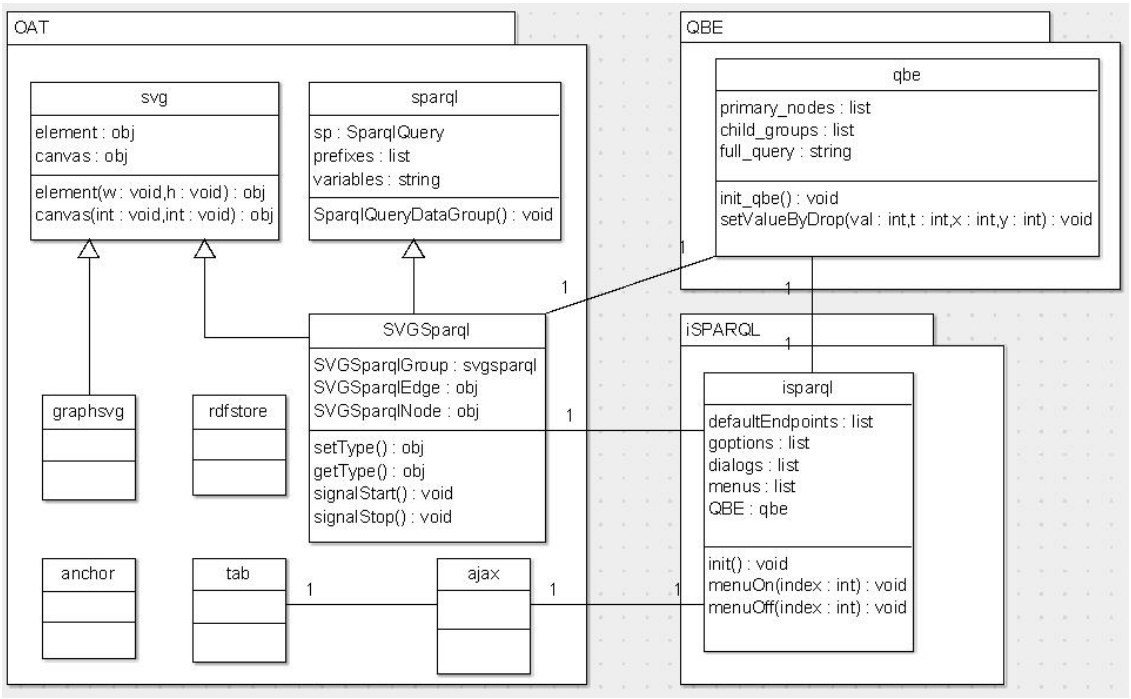
Paketstruktur von OAT und iSPARQL

Der Paketaufbau des Toolkits ist simpel gehalten. Die Ordner der ersten Ebene repräsentieren Komponenten des Toolkits, grundlegende Style- bzw. Dokumentdefinition oder stellen im Falle vom Ordner /oat/ die teils oben beschriebenen Grundfunktionalitäten in Form von JavaScript-Dateien bereit.

Im Ordner /isparql/ selbst kann die Demo-Version von iSPARQL über index.html getestet werden. Es liegen des weiteren die iSPARQL zu Grunde liegenden drei JavaScript-Dateien vor. Style-Dateien und die für die Query-Visualisierung notwendigen Bilder in separierten Ordnern runden die Komponente iSPARQL ab.

1.4 Mögliche Schnittstellen für den Graphical Query Builder

iSPARQL verwendet für den graphischen Query-Editor die Komponenten SVG, Sparql und die davon abgeleitete Komponente SVGSparql des OAT, wie im makroskopischen Paketdiagramm (siehe oben) erkennbar. Für das Praktikumsprojekt bietet es sich an, diese etablierten und ausgereiften Komponenten, die ihre Praxistauglichkeit bewiesen haben ebenfalls zu verwenden. SVG stellt hierbei eine SVG-„Zeichenfläche“, „Canvas“ genannt, zur Verfügung, die dann mit graphischen Repräsentationen von SPARQL-Objekten gefüllt werden können. Dafür lässt sich ebenfalls das SVGSparql-Objekt und dessen Methoden verwenden.



2 jQuery

2.1 Allgemeines

jQuery ist ein umfangreiches, freies Javascript-Framework, welches komfortable Funktionen zur DOM-Manipulation und -Navigation zur Verfügung stellt. Das von John Resig entwickelte Framework wurde im Januar 2006 auf dem BarCamp (NYC) in New York veröffentlicht und wird laufend weiter verbessert.

Es beinhaltet folgende Features: DOM-Selektierung und -Navigation (mit Unterstützung von CSS und einfachem XPath) DOM-Manipulation Effekte und Animationen Erweitertes Event-System AJAX-Funktionalitäten Hilfsfunktionen wie zum Beispiel die each-Funktion. Zahlreiche frei verfügbare Plugins Beliebige Erweiterbarkeit

2.2 Produktfunktionalitäten

2.2.1 Allgemein

jQuery kann folgendeweise in ein bestehendes HTML-Dokument eingebunden werden:

```
<script type="text/javascript" src="jQuery.js"></script>
```

Dann ist die Verwendung eines globalen Objektes namens „jQuery“ möglich (auch oft über die Variable „$“ als Shortcut). Die Verwendung von jQuery-Funktionen erfolgt stets über Zugriff auf dieses einzige globale Objekt. Beispielsweise kann die jQuery-Funktion „trim“, welche alle Leerzeichen vom Anfang und Ende einer Zeichenkette entfernt, folgendermaßen aufgerufen werden:

```
var testStr = "    Hallo!   ";
testStr = jQuery.trim ( testStr );   // Enthält nur noch "Hallo!"
```

I.A. wird statt „jQuery“ das Aliaszeichen „$“ geschrieben, was vielleicht komisch aussieht, aber den Code deutlich lesbarer macht. Obiges Beispiel könnte man also auch so schreiben:

```
var testStr = "    Hallo!   ";
testStr = $.trim ( testStr );   // Enthält nur noch "Hallo!"
```

2.2.2 Auswahl von DOM-Objekten

Die wichtigste Funktionalität von jQuery ist seine Fähigkeit, alle DOM-Objekte einer angegebenen Klasse als Menge aufzusammeln, deren Elemente dann alle gleichzeitig verarbeitet werden können. Will man z.B. die Menge aller Links (d.h. DOM-Objekte vom Typ „a“ : Anker, wie in „<a href=...“) erzeugen, geht es so:

```
jQuery("a");
```

bzw.

```
$("a");
```

Hierbei wurde also syntaktisch keine eigentliche Funktion aufgerufen (wie jQuery.trim() vorhin), sondern es wurde die Syntax „jQuery(<Klasse>)“ bzw. „$(<Klasse>)“ verwendet. Bei einem derartigen Aufruf bildet das jQuery-Objekt intern eine Liste aller zutreffenden DOM-Objekten der Webseite, die bis zum nächsten Aufruf dieser Form gespeichert wird. Man kann diese Funktionalität als SSelektion”bezeichnen, da eine Auswahl an Objekten erzeugt wird. Folgende Zeilen aus jQuery.js verdeutlichen die Funktionsweise, erstens die Definition von „jQuery“ und „$“:

```
jQuery = window.jQuery = window.$ = function( selector, context ) {
// The jQuery object is actually just the init constructor 'enhanced'
return new jQuery.fn.init( selector, context );
}
```

Der Aufruf wird also nur an „jQuery.fn.init“ weitergeleitet. Hier ein Ausschnitt von jQuery.fn.init:

```
jQuery.fn = jQuery.prototype = {
init: function( selector, context ) {

                //...

// Handle HTML strings
if ( typeof selector === "string" ) {
//...
                }

return this.setArray(jQuery.isArray( selector ) ?
selector :
jQuery.makeArray(selector));
}
```

Wird obige Funktion aufgerufen, so baut sie zunächst ein Array von gefundenen Objekten auf (dies wurde auskommentiert, da recht kompliziert), das dann mittels der Funktion „setArray“ in dem jQuery-Objekt abgespeichert wird:

```
// Force the current matched set of elements to become
// the specified array of elements (destroying the stack in the process)
// You should use pushStack() in order to do this, but maintain the stack
setArray: function( elems ) {
// Resetting the length to 0, then using the native Array push
// is a super-fast way to populate an object with array-like properties
this.length = 0;
Array.prototype.push.apply( this, elems );

return this;
}
```

D.h. also, daß das Objekt „jQuery“ auch selber ein Array ist. Man könnte z.B. durch

```
$("p")[0]
```

direkt auf den erstgefundenen Paragraphen des Dokuments zugreifen (wobei dieser erstgefundene Paragraph nicht unbedingt der eigentlich erste Paragraph des HTML-Dokuments sein muß). Somit erzeugt und speichert jQuery die gewünschte Liste von „Matches“ des Arguments - die Selektion.

Der Rückgabewert dieser „Select“-Funktion ist wieder das (einzige, globale) jQuery-Objekt selbst, was einen weitern jQuery-Aufruf in derselben Codezeile ermöglicht. Beispiel:

```
$("a").attr(href, "http://www.google.com");
```

Diese Zeile findet erst alle Links, dann setzt für jeden Einzelnen dessen Attribut „href“ auf „http://www.google.com external“. D.h. nach Ausführung dieser Zeile würden alle Links auf der Webseite nur noch nach „www.google.com“ führen. Diese Zeile ist nur deshalb gültig, weil der erste Teil $(ä”) das jQuery-Objekt zurückgibt, dessen Methode „attr“ dann aufgerufen wird. Der erste Teil der Zeile ($(ä”)) erzeugt eine Liste aller Links (intern vom jQuery-Objekt verwaltet), während der zweite Teil die Objekte dieser Liste manipuliert. Diese Struktur eines Aufrufs ist für jQuery ganz typisch. Man kann bei der Auswahl auch Benutzerdefinierte Klassennamen (etwa CSS) angeben, z.B. findet folgende Zeile alle DOM-Objekte, die als Klassenattribut den Wert testKlasse”besitzen:

```
$(".testKlasse");
```

Genaueres zum Auswählen von DOM-Objekten : http://docs.jquery.com/Selectors.

Hat man eine Menge von DOM-Objekten bereits ausgewählt, kann man diese Menge auch erweitern durch die „add“-Funktion:

```
$("a").add("p");
```

bildet die Menge bestehend aus allen Links und allen Paragraphen der Webseite. Durch Funktionen wie „next“, „parent“, „siblings“ können andere Mengen gebildet werden. Näheres: http://docs.jquery.com/Traversing

2.2.3 Manipulation von Objekten und sonstiges

Manipulation von ausgewählten Objekten ist möglich z.B. durch die bereits erwähnte „attr“-Funktion, oder auch „html”,’ „append“, „wrap“, „replaceAll“, „empty“, uvm.. Beispiel: Auswahl aller Unterobjekte von Objekten der Klasse „.klasse6“ und Hinzufügen von einem roten Hintergrundfeld:

```
<style>
  .redBG { background:red; }
</style>
$(".klasse6").contents().wrap("<div id="redBG"></div>")

Schauen wir uns die Funktion "wrap" in jQuery.js an:

wrap: function( html ) {
```

```
return this.each(function(){
jQuery( this ).wrapAll( html );
});
}
```

Diese Funktion iteriert also (mittels „each“, s.u.) über alle Elemente von „this“, d.h. über alle Elemente des jQuery-Objektes selber, das ja auch ein Array ist (s.o.). Auf jedes Element wird dann die Funktion „wrapAll“ angewandt: „wrapAll“ selber fügt dann jedem Element den HTML-Code hinzu. Da diese Funktion „wrapAll“ etwas komplizierter ist, wird sie hier nicht reproduziert, doch die Funktionsweise der Objektmanipulation dürfte dennoch klar sein.

Mehr zu Manipulation: http://docs.jquery.com/Manipulation

Durch jQuery.get(), jQuery.post() und jQuery.ajax() können HTTP-Requests ausgeführt werden, siehe http://docs.jquery.com/Ajax.

2.2.4 Chaining („The Magic of JQuery“)

-Aufrufe von Methoden geben eine neue jQuery-Instanz zurück, wenn man das JQuery-Objekt selbst verändert, z.B. durch Entfernen und, oder Hinzufügen von Elementen aus der Liste. Diese neue Instanz enthält wiederum eine Referenz auf die vorherige Instanz. Methoden-Aufrufe, die lediglich die gekapselten Elemente ändern, geben dieselbe Instanz zurück. Dadurch ist die Möglichkeit gegeben über Verkettungen den Code sehr kurz, übersichtlich und schnell zu halten. z.B.: `$('a').addClass('csAClass').show()`

Um innerhalb des „Chains“ außerdem noch die Menge der HTML-Objekte verändern zu können, gibt es weitere Funktionen innerhalb des „Chains“ (siehe „Traversierung“): z.B. : `$('#main').addClass('.main').find('a').addClass('csAClass');` (Hier bekommt jetzt das Objekt mit der ID „main“ auch die CSS-Klasse mainünd alle darin befindlichen a-Tags bekommen die Klasse „csAClass“.) -Methoden, die Werte aus den DOM-Elementen lesen oder berechnen, geben einen dem entsprechenden Wert zurück, z.B. einen String für den eingegebenen Wert eines Input-Elements oder eine Zahl für den Transparenz-Wert („opacity“) eines DIVs.

2.2.5 Hilfsfunktionen

Als letztes zu erwähnen ist noch die Hilffunktion jQuery.each(Objekt, Callback), die für jedes im ersten Parameter erhaltene Unterobjekt (der erste Parameter ist gewöhnlicherweise ein Array) die Funktion Callback”(2. Parameter) aufruft. Somit fungiert ëachïn etwa wie eine for-Schleife. Es folgt die Definition von ëachïn jQuery.js. Man sieht, daß lediglich eine Hilfsfunktion mit zusätzlichem Parameter aufgerufen wird:

```
each: function( callback, args ) {
return jQuery.each( this, callback, args );
}
```

Definition der Hilfsfunktion mit zusätzlichem Paramter:

```
// args is for internal usage only
each: function( object, callback, args ) {
var name, i = 0, length = object.length;
```

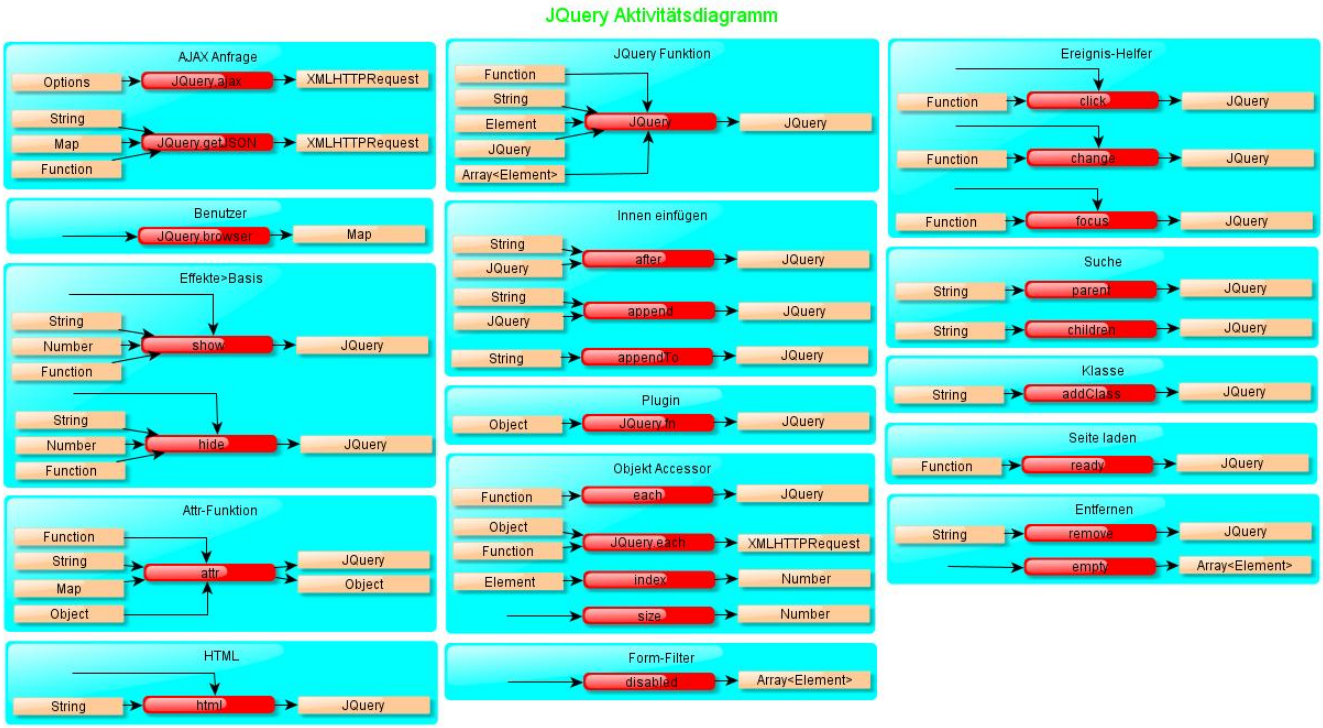
```
if ( args ) {
if ( length === undefined ) {
//...
} else
for ( ; i < length; )
if ( callback.apply( object[ i++ ], args ) === false )
break;

// A special, fast, case for the most common use of each
} else {
 //...

return object;
}
```

Klar dürfte sein, daß hier einfach nur eine for-Schleife enkapsuliert wurde. Siehe: http://docs.jquery.com/Utilities/jQuery.each#objectcallback

Man kann JQuery auch als eine Sammlung von Funktionen auffassen. Dahingehend ist eine detailierte Gliederung der einzelnen Funktionen in folgende Funktionsgruppen möglich :



3 JavaScript

3.1 Prototypen

Es ist möglich in JavaScrip bestehende Methoden bzw. Objekte nachträglich zu beeinflussen. Dies funktioniert mittels sogenannten Prototypen auf die JavaScript basiert. Prototypen sind selbst Objekte, die als Vorlage zur Erzeugung weitere Objekte verwendet werden. Dabei beinhaltet der Prototyp alle Eigenschaften, die Initial zum Objekt gehören. Dadurch kann jedem Objekt ein Prototyp zugeordnet werden, dessen Eigenschaften das Objekt mit übernimmt. Durch die Prototypen-Funktionen ist es möglich Vererbung in JavaScript zu benutzen. Mit der Objekteigenschaft prototype kann man neue Eigenschaften und Methoden einem Objekt zuweisen.

Ein Code-Beispiel:

```
function myClass(myVar){
   this.infos = myVar;
   this.getInfos;
}
myClass.prototype.getInfos = function(){
   return this.infos;
}
```

Die Klasse myClass ist „eigentlich“ als Funktion definiert, kann aber mit Member-Funktionen überladen werden und so als Klasse fungieren. Für Member-Funktionen- und Variablen wird das Schlüsselwort this”vorangestellt.

Auch jQuery wird über prototype erweitert: Beim Aufruf der funktion jQuery.fn(newFunc) wird newFunc an den Prototyp weiter gegeben und dort gespeichert.

Ein Beispiel: Zeile 35 des jQuery-Quellcodes

```
jQuery.fn = jQuery.prototype [...];
```

somit kann jQuery wie folgt erweitert werden

```
jQuery.fn.extend({
  checkAll: function() {
    return this.each(function() { this.checked = true; });
  },
  uncheckAll: function() {
    return this.each(function() { this.checked = false; });
  }
});
```

3.2 Global Namespace Pollution

Unter Global Namespace Pollution versteht man das Problem, dass es viele Funktionen und Variablen global definiert werden, deren Namen aber nicht immer eindeutig sind. Global Namespace Pollution ist ein zu vermeidendes Phänomen, da ein Programmierer, der sich mit so einer Applikation zu beschäftigt, viel Zeit und Energie darauf verwenden muss, sich die reservierten Namen zu merken und eindeutige Namen für neue Symbole zu finden. Zusätzlich können Namensraum-Kollisionen verschiedenartige Probleme verursachen, die teilweise sehr schwer zu lokalisieren und aufwändig zu beheben sind.

jQuery bietet nur ein einziges globales Objekt an, das zentral alle Methoden bereithält („jQuery“ bzw. „$“).