# Nacharbeitung Meilenstein 1

Anmerkung: Aus persönlichen Gründen war es Siegfried leider nicht möglich, dieses Dokument in der gewohnten, schönen LaTeX-Form zu erstellen. Dies wird dann zeitnah nachgeholt. Wir nehmen an, dass dies kein Problem darstellt, da es ja ein Nachtrag zu unserem Vortrag vom 28. Januar 2014 darstellt. Die Inhalte dieses Dokuments fließen natürlich mit in den Projektvertrag ein.


Daten- /Endpointqualität
Gestaltung von Dummydaten
Caching von Daten
Ausgabe der Resultate von SPARQL-Anfragen
Speichern von GSB-Anfragen
JSON vs. XML
Definition der GSBL


## Daten- /Endpointqualität

Um mit dem GSB kompatibel zu sein, sollten die folgenden Merkmale durch den Endpoint gewährleistet werden:

Die Property **rdf:type** beschreibt, von welcher Klasse die betreffende Ressource eine Instanz darstellt. Dies sind, ausgehend von den W3C-Standards zu rdf, rdfs und owl, unter anderem *owl:Class*, *owl:DatatypeProperty* und *owl:ObjectProperty*. Diese Unterscheidung ist für den GSB insofern von hoher Bedeutung, da sie auch in der Sprachdefinition eine entscheidende Rolle spielt. Es ist jedoch denkbar, auch ohne vorhandenen **type** auszukommen, unter Voraussetzung der Properties **domain** und **range**: hierbei kann die **range** auf Datentypen wie *xsd:string* oder *xsd:double* geprüft werden, um die Ressource als *DatatypeProperty* zu identifizieren, oder sie ansonsten als *ObjectProperty* zu behandeln. Damit könnte ein Datenbestand vom GSB auch ohne **type** erfasst werden.

Daraus ergibt sich, dass die Properties **rdfs:domain** und **rdfs:range** jeweils definiert sein müssen. Ohne eine festgelegte **domain** können die möglichen Eigenschaften einer Klasse nicht ohne weiteres ermittelt werden, bei einer fehlenden **range** die passende Zielklasse nicht.

Die Properties **rdfs:subClassOf** bzw. **rdfs:subPropertyOf** sind äußerst nützlich, um Vererbungen zwischen Klassen bzw. Eigenschaften zu finden. Denkbar wäre etwa eine *Schauspieler*-Klasse, die gleichzeitig eine *Person* ist, oder eine *trittAufIn*-Eigenschaft, welche die selben Ressourcen verknüpft wie eine *RolleIn*-Eigenschaft. Doch unter der Voraussetzung der vorhergehenden Properties, welche die relevanten Verknüpfungen einer Klasse abdecken, können sie als nicht zwingend notwendig angesehen werden.

Desweiteren sind die Properties **rdfs:label** bzw. **rdfs:comment** als weitestgehend optional anzusehen. Sie erhöhen die Nutzerfreundlichkeit durch natürlichsprachliche Bezeichnungen

bzw. Beschreibungen einer Ressource, welche sich ansonsten eventuell nur mit mehr oder weniger kryptischen Bezeichnern identifizieren - doch dies kann und muss im Zweifelsfall ausreichen, da es die Benutzung der Anwendung nur bedingt beeinträchtigt.

Für die Hintergrundkommunikation, um zum Beispiel Klassen oder Eigenschaften der selbigen zu laden, sollte als Rückgabeformat JSON zur Verfügung stehen, da JSON-Objekte sich leicht in Javascript auslesen und manipulieren lassen (bzw. ein dementsprechend leicht zu konvertierendes Format).

[http://www.w3.org/TR/owl-features/]
[http://www.w3.org/TR/rdf-schema/]

## Gestaltung von Dummydaten

Da die Gestaltung einer funktionalen graphischen Oberfläche das Hauptziel des GSB ist, tritt die Bindung an einen realen Endpoint in den Anfängen des Projekts in den Hintergrund. Es sollen aber schon Dummydaten als JSON-Objekte hinterlegt werden, da Ergebnisse von SPARQL-Anfragen als JSON-Objekte serialisiert werden können.
[http://www.w3.org/TR/rdf-sparql-json-res/]

## Caching von Daten

Da das Caching von uns als Kannziel definiert wurde, sei nur kurz angemerkt: Falls ein Caching implementiert wird, sollte es für den Administrator (evtl. auch für den Endnutzer) ausschaltbar sein, da eventuell der Endpoint (oder eine Zwischeninstanz) Caching betreibt. Ein implementiertes Caching wäre wahrscheinlich nur beim Endnutzer (Stichwort: HTML Local Storage) angedacht, um kurzzeitig zum Beispiel den Klassennamen und Eigenschaften von Klassen zu speichern. Dies könnte die Performanz auf Seiten des Endnutzers erhöhen.

## Ausgabe der Resultate von SPARQL-Anfragen

Generell wird dem Nutzer die Möglichkeit gegeben die Resultate in den Formaten zu speichern, die der Endpoint bereitstellt. Als vom GSB definierte Resultatanzeige ist eine verbesserte HTML-Tabelle mit Features wie zum Beispiel Zeilensortierung durch Javascript als Kannziel denkbar.

## Speichern von GSB-Anfragen

Als Kannziel wurde definiert, dass man GSB-Anfragen als JSON-Objekt speichern kann. Dies könnte dafür verwendet werden um
a) dem Nutzer die Anfrage erneut ohne Wiederherstellung der GSBL zu ermöglichen
b) die GSBL wiederherzustellen und somit auch Beispielanfragen oder das Verteilen von Anfragen zu erlauben.
Ein noch tiefergreifendes Kannziel wäre die Konvertierung von SPARQL-Anfragen zu GSBL.

## JSON vs. XML

Beide Sprachen bieten ein hohes Maß an Simplizität, Offenheit und Interoperabilität.
JSON ist jedoch XML dank seiner für den GSB zweckdienlicheren Ausstattung vorzuziehen.

Anders als bei XML, bei der die Daten in einer Baumstruktur gespeichert werden, werden die Daten in JSON in einem Array oder Objekt gespeichert, wodurch die Übertragung der Daten zu objektorientierten Sprachen wie JavaScript stark vereinfacht wird.
Des Weiteren bietet die einfacher gestaltete Syntax eine höhere Übersichtlichkeit, wodurch es leichter bearbeitet und gelesen werden kann.

## Definition der GSBL

Die GSBL soll möglichst genau definiert werden. Dies geschieht zum großen Teil unseres Vorprojekts, aber wahrscheinlich begleitet uns die Ausarbeitung der Definition bis zum Ende. Zu diesem Zeitpunkt (3. Februar 2014) gibt es dementsprechend noch keine komplett ausgereifte Definition. Um unsere Fortschritte auf diesem Gebiet nachvollziehbar zu machen, gibt es im Anhang dieses Dokuments einen Entwurf der GSBL-Definition vom Stand des 3. Februar 2014.