BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

# Konzeption und Implementierung eines Kompetenz-Verzeichnis für digitale Zertifikate auf Basis von Linked-Data.

## Bachelorarbeit

zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)
im Fach Medieninformatik


eingereicht von
Christian Mehns

| | |
|---|---|
| Studiengang | Medieninformatik |
| Fachbereich | FB VI - Informatik und Medien |
| Mat. Nr. | 791881 |

**Erstgutachter:** Prof. Dr.-Ing. Johannes Konert
Beuth Hochschule für Technik Berlin

**Zweitgutachter:** Prof. Dr. Görlich
Beuth Hochschule für Technik Berlin


Berlin, 04. Oktober 2016

# Inhaltsverzeichnis

# 1. Einleitung

Die globale Vernetzung hat in den letzten Jahren durch da Internet enorm zugenommen. Wir benutzen modernste Technologie, um miteinander zu kommunizieren, uns Wissen anzueignen oder eine Arbeitsstelle zu finden. Der Bildungsbereich ist in seiner Form immer noch sehr traditionell. Was den Nachweis von Lernerfolgen und Kompetenzen betrifft, ist ein Zeugnis als Zusammenfassung aller Bildungsleistungen immer noch der Standard. In einer globalisierten Welt in der Studenten und Angestellte sich über Grenzen hinweg bewegen und eine Akkreditierung für ihre Zeugnisse, Fähigkeiten, Wissen und Kompetenzen benötigen, die sie zum Beispiel in einem anderen Land, mit anderer Sprache, erworben haben, rückt das Problem der Vergleichbarkeit immer mehr in den Vordergrund.[1]

Eine möglichen Ansatz, zur Lösung dieser Anforderungen, bieten dabei die digitalen Lern-Zertifikate, auch als digitale Abzeichen (engl. *digital badges*) bezeichnet. Ein Abzeichen (engl. *Badge*) stellt ein Symbol oder Indikator für eine Leistung, Fähigkeit, Qualität oder Teilnahme dar. Digitale Badges sind also auch als eine Online Repräsentation einer Fähigkeit, die man sich angeeignet hat, einsetzbar. Der Lernerfolg eines Individuums lässt sich damit wesentlich detaillierter abbilden als ein Zeugnis, und beinhaltet dabei nicht nur Nachweise für formelle Lernergebnisse, wie ein Abschlusszeugnis einer Schule oder Universität, sondern auch informelle Lernergebnisse, die z.B. durch einen Online Kurs erworben wurden. Neben dem Nachweis einer Leistung sollen Badges den Lernenden auch motivieren, sich ein bestimmtes Wissen anzueignen oder eine Fähigkeit zu erwerben.[2]

Als frei verwendbares, offenes Format, ist mittlerweile der Open Badges Standard im Bereich der digitalen Abzeichen sehr populär geworden. Mit diesem Standard ist es prinzipiell möglich, alle denkbaren Formen von Lernerfolgen und Kompetenzen abzubilden.

[1] Vgl. [Kon16], Competency Frameworks and Standardized Competencies, S.5
[2] Vgl. Badges, https://wiki.mozilla.org/Badges (10.08.2016).

## 1.1 Problemstellung

Ein Problem, das die Verbreitung des Open Badges Standard, im formalen Bildungsbereich und im Bereich des Personalmanagement, behindert, ist zum einen, die fehlende Eindeutigkeit von Kompetenzdefinitionen und zum anderen die mangelnde Vergleichbarkeit der, von Badges abgebildeten Kompetenzen. Da es sich um einen offenen Standard handelt, und jede Organisation oder Einzelperson ihre eigenen Badges ausstellen kann, unterliegen auch die Kompetenzdefinitionen keinem Standard, sondern werden gegebenenfalls bei jedem Badge neu definiert. Dies führt zu einer Fülle an Kompetenzdefinitionen, die sich im Grunde alle auf eine gleichbedeutende Kompetenz beziehen können. Eine Lösung wäre ein zentrales Verzeichnis, das alle möglichen standardisierten Kompetenzen enthält und problemlos auffindbar macht. Jedes Badge könnte dann auf so eine standardisierte Kompetenz verweisen und eine Eindeutigkeit wäre gewährleistet.

Das Problem der Vergleichbarkeit könnte hiermit auch gehandhabt werden. Momentan ist er schwer möglich die Kompetenzen, die ein bestimmtes Badge darstellt mit einem anderen Badge zu vergleichen, was auch daher rührt, dass ein Badge sich jeweils auf unterschiedliche Kompetenzen beziehen kann. Würden die Informationen, die einem Kompetenz-Rahmenwerk zugrunde liegen, in einer semantischen Form vorliegen, so könnten auch Metadaten über die Beziehungen zwischen verschiedenen Kompetenzen gespeichert werden, wie zum Beispiel, dass eine bestimmte Kompetenz genau einer anderen entspricht. Diese Beziehungen können dann von einem Computer automatisch verarbeitet werden.

Im Bildungs- und Business-Bereich haben sich bereits verschiedene, spezifische Rahmenwerke (engl. *frameworks*) entwickelt und Konvergenzen bzw. Ähnlichkeiten zwischen einigen von ihnen wurden definiert. Beispiele für solche Rahmenwerke sind DQR[3], ECVET[4] und e-CF[5]. Prinzipiell hat jeder Mensch Zugriff auf die Dokumente, in denen die Rahmenwerke definiert werden, allerdings kann ein Computer die enthaltenen Informationen nicht ohne weiteres verarbeiten. Um das zu ermöglichen, müssten die Daten in einer definierten, semantischen Form vorliegen. Es wäre Computersystemen somit möglich, Konvergenzen von definierten Badges und die Abhängigkeiten von bestimmten Kompetenzen festzustellen und zu verarbeiten.

[3] Vgl. DQR - Deutscher Qualifikationsrahmen für lebenslanges Lernen, http://www.dqr.de/ (20.07.2016).
[4] Vgl. ECVET - European Credit System for Vocational Education and Training, http://www.ecvet-info.de/ (20.07.2016).
[5] Vgl. e-CF - European e-Competence Framework, http://www.ecompetences.eu (20.07.2016).

## 1.2 Ziel und Aufbau der Arbeit

Die Entwicklung eines digitalen Verzeichnisses für Kompetenzen ist ein Schritt, um dem, bereits erwähnten Problem der Vergleichbarkeit zu begegnen. Der Vorschlag für eine solche Implementierung wurde auch in der Veröffentlichung *Outcome 03-A2 - Competency Directory*[6] gemacht, die im Rahmen des *Open Badge Networks*[7] entstanden ist. Dabei wurde analysiert, wie Open Badges als Zertifikate für gezeigte Kompetenzen dienen und dabei auf Kompetenzdefinitionen innerhalb standardisierter Kompetenz-Rahmenwerke verweisen können. Der Open Badge Standard soll im nächsten Kapitel kurz näher vorgestellt werden und aufgezeigt werden, wie auf standardisierte Kompetenzdefinitionen verlinkt werden kann.

In dem Paper wird des weiteren das *InLOC* Informationsmodell vorgeschlagen, um verschiedenste Kompetenz-Rahmenwerke abbilden zu können und in maschinen-lesbarer Form zur Verfügung zu stellen. Rahmenwerke, die in dieser Form ausgezeichnet sind, könnten dann automatisch in das Verzeichnis eingelesen werden und würden ihre Semantik behalten. Um abzuwägen, ob dieser Vorschlag in dieser Form umsetzbar ist, sollen die Themen semantisches Web (engl. *semantic web*) und verlinkte Daten (engl. *linked data*) näher betrachtet werden, sowie das InLOC Modell und der Open Badge Standard analysiert werden. Dabei sollen auch alternative Ansätze diskutiert werden.

Um eine Abwägung einer Umsetzung dieses Vorschlags vorzunehmen, sollen sowohl das InLOC Informationsmodell näher betrachtet werden, als auch die Themen *semantisches Web* (engl. *semantic web*) und *verlinkte Daten* (engl. *linked data*).

Nach der Auseinandersetzung mit diesen Grundlagen soll das, als Lösung vorgeschlagene, digitale Kompetenz-Verzeichnis entworfen werden. Dafür werden zunächst die Anforderungen analysiert und ein Entwurf für einen Prototypen entwickelt. Danach folgt ein Kapitel über Details zur Umsetzung.In Kapitel sechs soll der Prototyp evaluiert werden und Schwierigkeiten in der Entwicklung aufgezeigt werden. Im letzten Kapitel wird dann ein Fazit gezogen und ein Blick auf eine mögliche Weiterentwicklung des Prototypen geworfen.

Ein Prototyp für ein solches digitales Verzeichnis auf Basis semantischer, verlinkter Daten, soll im Rahmen dieser Bachelorarbeit entworfen und implementiert werden.

[6] Vgl. [Kon16], Proposed solution concept, S.10.
[7] Vgl. Open Badge Network http://www.openbadgenetwork.com/, (10.08.2016).

# 2. Grundlagen

In diesem Kapitel sollen zunächst die Themen semantisches Web und Linked Data vorgestellt werden und danach der Open Badges Standard und das InLOC Informationsmodell. Diese Themen sollen als Grundlagen für den Entwurf und die Implementierung des Prototypen dienen.

## 2.1 Semantic Web und Linked Data

Die Probleme des Internets und die Lösung durch das Konzept des semantischen Webs, wurde in dem Buch "Semantic Web" [Hit08] analysiert. Dieses Kapitel nimmt auf die im Buch vorgestellten Konzepte und Ideen Bezug.

### 2.1.1 Probleme des Internets

Das Problem an der zunehmenden Menge an Informationen, die das Internet repräsentiert, ist die Ausrichtung jener Repräsentation auf den Menschen als Endnutzer. Der Mensch als Benutzer kann problemlos die Informationen auf Webseiten erfassen, in andere Darstellungsformen transformieren und zu anderen Informationen in Beziehung setzen. Eine Maschine hingegen kann dies in der Regel nicht leisten. Daraus ergibt sich das Problem, dass bestimmte Informationen nur schwer gefunden werden können. Suchmaschinen wie Google können dieses Problem zwar in Ansätzen lösen, allerdings hauptsächlich durch statistische Methoden und die Lokalisierung von Zeichenketten im Text. Besser wären hier aber weniger Stichwort basierte sondern mehr inhaltliche, semantische Suchmaschinen. Eine Suche, die die Bedeutung von Inhalten interpretieren und verarbeiten kann.

Weitere Probleme ergeben sich durch die dezentrale Struktur und Organisation des Internets, die zwangsläufig zu einer Heterogenität der vorhandenen Informationen führen. Unterschiedliche Dateiformate bzw. Kodierungstechniken, die verschiedenen Sprachen bis hin zum unterschiedlichen Aufbau vieler Internetseiten machen es fast unmöglich über das Web verteilte Informationen zu bündeln, einheitlich aufzubereiten und weiterzuverwenden. Hier haben wir es mit einem Problem der *Informationsintegration* zu tun. Schließlich wäre es denkbar, dass ein Nutzer nach einer Information sucht, die so vielleicht gar nicht im Internet zu finden ist, sondern lediglich aus einer Reihe verschiedener Fakten, die im Web verteilt sind, zu schlussfolgern ist. Beispielsweise ergibt die Information, dass San Fernando in Venezuela und Berlin in Deutschland liegt, gemeinsam mit der Information, dass die Differenz der lokalen Zeiten fünf Stunden beträgt, dass man aus Berlin möglichst nicht vor der Mittagszeit in San Fernando anrufen sollte. Hierbei handelt es sich um die Problematik des *impliziten Wissens*.[8]

[8] Vgl. [Hit08] Kapitel 1.3, Probleme des Web, S.10 f.

### 2.1.2 Das Semantische Web

Zur Lösung dieser Probleme kann man grundsätzlich zwei komplementäre Herangehensweisen unterscheiden. Einerseits kann man Methoden der Künstlichen Intelligenz anwenden, die die kognitiven Aufgaben, die ein Mensch bei der Verarbeitung von Informationen erfüllen würde, übernehmen und die primär für den Mensch repräsentierten Daten in maschinenlesbare Daten umwandelt. Eine alternative Herangehensweise besteht im Ansatz des Semantic Web. Das Semantic Web steht dabei für die Idee, die Informationen schon von vornherein so zur Verfügung zu stellen, dass sie für Maschinen lesbar und vor allem interpretierbar sind. Während also das World Wide Web eine Möglichkeit darstellt alle Daten der Welt miteinander zu vernetzen, zeigt das Semantic Web einen Weg auf, um die Informationen der Welt auf der Ebene ihrer Bedeutung miteinander zu verknüpfen. Es werden dabei unstrukturierte in strukturierte Daten umgewandelt.

Als Grundvoraussetzung dafür ist es erforderlich einheitliche und offene Standards für die Beschreibung von Informationen zu vereinbaren, die es letztendlich ermöglichen sollen, Informationen verschiedener Anwendungen und Plattformen auszutauschen und zueinander in Beziehung zu setzen. Diese Fähigkeit wird als Interoperabilität bezeichnet. Dabei ist es nicht nur wichtig, dass die Standards formal klar definiert sind, sondern dass sie auch flexibel und erweiterbar sind, ohne später revidiert werden zu müssen. Die Ausarbeitung und Beschreibung solcher Standards hat sich das *World Wide Web Consortium* (W3C)[9] zur Aufgabe gemacht. Gegründet vom Internet-Erfinder Tim Berners-Lee, definiert das W3C seit 1994 neben Standards zu HTML, XML, XHTML, CSS und weiteren, für das Internet standardisierte Techniken, auch grundlegende Standards für das *Resource-Description- Framework* (RDF) und die *Web Ontology Language* (OWL) . Letztere sind Informations- Spezifikationssprachen bzw. Ontologiesprachen, die speziell für die Verwendung im Semantic Web entwickelt wurden.[10]

### 2.1.3 Syntax und Semantik

Während man unter Syntax eine Menge von Regeln zur Strukturierung von Zeichen und Zeichenketten versteht, steht der Begriff Semantik allgemein für die Bedeutung von Wörtern, Phrasen oder Symbolen. Im Bereich der Informatik versteht man unter Semantik die Bedeutung von Worten bzw. Zeichen (-ketten) und ihre Beziehung untereinander.[11]

[9] Vgl. W3C https://www.w3.org/ (13.08.2016)
[10] Vgl. [Hit08] Kapitel 1.3, Das Semantic Web, S.11
[11] Vgl. [Hit08] Syntax vs. Semantik, S.13

### 2.1.4 URI

Eine URI (Universal Resource Identifiers) ist ein universell einzigartiger Name, der benutzt wird, um Dinge im semantischen Web zu benennen. URIs stellen eine verallgemeinerte Version von URLs (Uniform Resource Locator) dar. Während URLs zur eindeutigen Lokalisierung von Webseiten benutzt wird, sodass jeder Nutzer, der die URL in einen Browser eingibt, bei dem selben Dokument landet, benutzt das semantische Web URIs, um eine eindeutige Bedeutung mit einem Namen zu verknüpfen. Die internationalisierte Form der URI ist die IRI (Internationalized Resource Identifier). IRIs erweitern die erlaubten Zeichen in URIs zu fast allen Zeichen des Universal Character Set (Unicode/ISO 10646).[12]

### 2.1.5 RDF

Dieser Abschnitt bezieht sich auf das Kapitel 3, des Buches "Semantic Web" [Hit08] S 35, ff.
Das Resource Description Framework (RDF) selbst ist kein Datenformat, sondern eine formale Sprache für die Beschreibung strukturierter Informationen. Es soll Anwendungen ermöglicht werden, Daten im Web auszutauschen, ohne dass ihre ursprüngliche Bedeutung dabei verloren geht. Es geht, im Gegensatz zu HTML und XML nicht um einer korrekte Darstellung von Dokumenten, sondern auch um die Kombination und Weiterverarbeitung der enthaltenen Informationen. RDF wird als das grundlegende Darstellungsformat für die Entwicklung des Semantischen Web gesehen.


Abbildung 1: Beispiel eines RDF-Graphen

Ein RDF-Dokument beschreibt einen gerichteten Graphen, also eine Menge von Knoten, die durch gerichtete Kanten verbunden werden. Knoten und Kanten sind dabei jeweils mit eindeutigen Bezeichnern (URIs) beschriftet. XML wird im Gegensatz mit Hilfe von Baumstrukturen dargestellt. RDF wurde nicht für die hierarchische Strukturierung einzelner Dokumente entwickelt, sondern für die Beschreibung von allgemeinen Beziehungen zwischen Ressourcen. Abbildung 1 zeigt ein einfaches Beispiel eines Graphen aus zwei Knoten und einer Kante, die Beziehung zwischen Buch und Verlag ist dabei eine Information, die keinem der beiden Knoten klar untergeordnet ist. Diese Struktur aus Subjekt, Prädikat und Objekt wird als Tupel bezeichnet. In RDF werden daher die Beziehungen zwischen zwei Knoten als grundlegende Informationseinheiten betrachtet.

[12] Vgl. https://de.wikipedia.org/wiki/Internationalized_Resource_Identifier (20.08.2016)

Ein Grund, weshalb RDF Graphen benutzt, ist, dass RDF nicht für die hierarchische Strukturierung einzelner Dokumente entwickelt wurde, sondern für die Beschreibung von allgemeinen Beziehungen zwischen Ressourcen. RDF wurde außerdem als Beschreibungssprache für Daten im World Wide Web und anderen elektronischen Netzen konzipiert, in dieser Umgebung werden Daten dezentral verwaltet und gespeichert. Für RDF ist es kein Problem, Informationen aus vielen Quellen zu kombinieren: Würden die Daten in der Webseite als XML-Strukturen vorliegen, wäre eine Zusammenführung wesentlich komplizierter. Graphen in RDF sind also für die Komposition dezentraler Informationen geeignet. RDF verwendet grundsätzlich URIs zur Bezeichnung aller Ressourcen, um diese eindeutig identifizieren zu können.

### 2.1.6 Linked Data

Während der Begriff *Semantisches Web* eher ein theoretisches Konzept beschreibt, kann der Begriff Linked Data als eine konkrete Anwendung von semantischen Technologien betrachtet werden.

Beim traditionellen Hypertext Web stehen die Verbindungen, also Links, alle für die gleiche Bedeutung: Hinter dem Link sind weiterführende Informationen verborgen. Linked Data erweitert das "Dokumenten Web" dahingehend, als dass die Verbindung für alles Mögliche stehen kann, das man mit Hilfe einer URI benennen kann. Hyperlinks können auf diese Weise verschiedenste Bedeutungen annehmen und beschreiben.[13]

Linked Data ermöglicht es, direkt auf die Daten in einem Dokument zu verlinken anstatt nur auf das Dokument selbst. Das Konzept stellt einen universellen Weg zur Verfügung, um Daten im Web zu verbreiten und sie für den Computer verarbeitbar zu machen. Der Begriff Linked Data bezieht sich auf eine Reihe von bewährten Verfahren für die Veröffentlichung und Verbindung von strukturierten Daten im Web, unter der Benutzung von internationalen Standards des World Wide Web Consortiums (W3C).[14]

Tim Berners-Lee hat vier Linked Data Prinzipien[15] festgelegt, um die Voraussetzungen für Linked Data zu definieren und einen gewissen Standard festzulegen. Dazu gehören die folgenden Punkte:

1. Für die Benennung von Dingen sollen URIs verwendet werden.
2. Es sollen konkret HTTP URIs verwendet werden, damit ein Benutzer den Namen nachschlagen kann.
3. Folgt ein Benutzer einer URI, so soll er nützliche Zusatzinformationen vorfinden, in Form von Standards wie RDF und SPARQL.
4. Einbindung von Verlinkungen zu anderen URIs, damit der Benutzer weitere Dinge entdecken kann.

[13] Vgl. [Woo14] Vorwort, S. XIII
[14] Vgl. [Woo14] Kapitel 1, Introducing Linked Data, S.3
[15] Vgl. Linked Data, https://www.w3.org/DesignIssues/LinkedData.html (16.08.2016)

Linked Data benutzt RDF als Datenmodell und repräsentiert dieses in einer von mehreren Syntaxen, wie RDFa oder JSON-LD. Die Entwickler des InLOC Standards empfehlen die Verwendung von XML oder JSON, bzw. JSON-LD.[16] Da XML zunehmend von JSON abgelöst wird, soll JSON-LD nun näher betrachtet werden.

### 2.1.7 JSON-LD

JavaScript Object Notation, kurz JSON, ist ein kompaktes Datenformat für die Speicherung und Serialisation strukturierter Daten und den Austausch dieser Daten zwischen Anwendungen. JSON ist für Menschen einfach zu lesen und zu schreiben und für Maschinen einfach zu parsen und zu generieren.[17]

JSON-LD ist eine vom W3C veröffentlichte Spezifikation[18], die ein JSON-basiertes Format für serialisierte verknüpfte Daten (Linked Data), darstellt. Webservices und Web- applikationen, die ihre Daten bevorzugt in JSON austauschen, finden mit diesem Format leichten Anschluss zum Semantischen Web und können besser zusammenarbeiten. Es handelt sich bei einem JSON-LD Objekt um ein RDF Serialisationsformat, das korrekter RDF Syntax entspricht und gleichzeitig ein gültiges JSON Objekt ist.[19]

```
{
  "@context":
  {
    "name": "http://xmlns.com/foaf/0.1/name",
    "depiction":
    {
      "@id": "http://xmlns.com/foaf/0.1/depiction",
      "@type": "@id"
    },
    "homepage":
    {
      "@id": "http://xmlns.com/foaf/0.1/homepage",
      "@type": "@id"
    },
  },
  "name": "Manu Sporny",
  "homepage": "http://manu.sporny.org/",
  "depiction": "http://twitter.com/account/profile_image/manusporny"
}
```

Listing 1: Beispiel eines JSON-LD Dokuments

[16] Vgl. InLOC Technical Bindings, http://www.cetis.org.uk/inloc/Bindings (13.08.2016).
[17] Vgl. Einführung in JSON, http://json.org/json-de.html (13.08.2016).
[18] Vgl. JSON-LD 1.0, A JSON-based Serialization for linked Data https://www.w3.org/TR/2014/REC-json-ld-20140116/ (13.08.2016).
[19] Vgl. [Woo14] Kapitel 2.4.4 JSON-LD--RDF for JavaScript Developers, S.52

JSON-LD verwendet, im Gegensatz zu JSON, global eindeutige Bezeichnungen (URIs) als Eigenschaftsbezeichner. Damit wird gewährleistet, dass jede Angabe eines Eintrags einem standardisierten Vokabular entspricht.

Listing 1 zeigt ein Beispiel für ein JSON-LD Dokument, auffällig sind dabei vor allem, die mit einem @ annotierten, Schlüsselwörter *context, id* und *type*. Sie sind mit einer definierten Semantik belegt. Der mit *@context* annotierte Kontext-Teil, verlinkt Objekteigenschaften in einem JSON-Dokument mit einem Konzept innerhalb einer Ontologie, dabei wird mit @id eine URI festgelegt und mit @type der Typ der Eigenschaft. Die im Kontext definierten Abkürzungen können dann im eigentlichen Objekt benutzt werden, ohne die URI angeben zu müssen. Neben context, type und id existieren 10 weitere Schlüsselwörter, die alle mit einem @ annotiert werden.[20]

### 2.1.8 Strukturierte Daten & Semantische Annotation

Die Abbildung 2 zeigt das Ökosystem strukturierter Daten. Linked Data, als konkrete Anwendung des Semantischen Webs, benutzt das RDF Datenmodell, um semantische Daten abzubilden. Um diese Daten konkret zu speichern und auszutauschen, braucht es ein RDF-konformes Format. Neben dem bereits vorgestellten JSON-LD, das vor allem für die Serialisierung und den Austausch von semantischen Daten geeignet ist, gibt es auch Formate, die eine Einbettung von semantischen Daten direkt in ein HTML-Dokument erlauben. In dieser Form werden die Informationen als *strukturierte Daten* (engl. *structured data*) bezeichnet. Die Formate basieren auf dem Konzept der *Semantischen Annotation*.

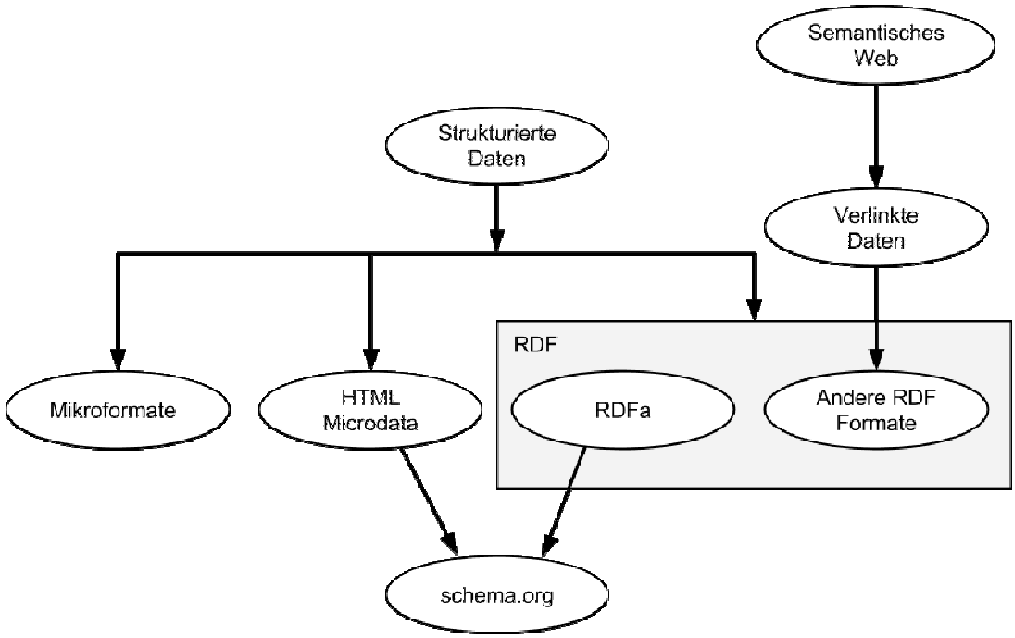


Abbildung 2: Das Ökosystem strukturierter Daten[21]

[20] Vgl. 1.2 Syntax Tokens and Keywords, http://json-ld.org/spec/ED/json-ld-syntax/20120522/ , (20.08. 2016)

[21] [Woo14] S. 242 Abbildung 11.3, vom Autor übersetzt.

Die semantische Annotation von Text, ist ein Prozess, bei dem Textfragmente zu strukturierten Informationen verbunden werden. Diese strukturierten Informationen (engl. *structured data*) können alle möglichen Begriffe und Konzepte repräsentieren, wie zum Beispiel Organisationen, Orte, Produkte oder Krankheiten. Beim Prinzip der semantischen Annotation wird die Extraktion von Informationen (engl. *text mining*) mit semantischen Technologien kombiniert.

Die Qualität des Textanalyseprozesses wird durch die Verwendung von Ontologien aus externen Wissensbasen und strukturierten Datenquellen (z.B. Schema.org) verbessert. Im Unterschied zum Prozess des Verschlagwortens (engl. *tagging*), wird bei der semantischen Annotation eine eindeutige (globale) Referenz, für eine, im Text erkannte, Entität gefunden. Bei der semantischen Suche werden über diese Methode die verschiedenen Bedeutungen geliefert. Es führt außerdem zu einer verbesserten Datenintegration, da Dokumente von verschiedenen Datenquellen, ein identisches semantisches Konzept haben können.

### 2.1.9 Formate zur semantischen Annotation

Um die semantische Annotation anzuwenden, braucht es einen eindeutig definierten Mechanismus, der in Form verschiedener Formate bereitgestellt wird. Die drei am weitesten verbreiteten Formate[22] sind dabei Mikroformate[23] (engl. *microformats*), HTML Microdata[24] und RDFa[25]. Alle drei Formate sind RDF konform und können somit dieselben Aufgaben erfüllen, Unterschiede gibt es nur im Detail. Die Autoren des InLOC Modells schlagen RDFa als Format vor, um die strukturierten Daten, die das Informationsmodell abbilden, in eine Webseite einzubinden.

### 2.1.10 Vokabular

Ein Vokabular liefert ein Schema für das Linked Data Web, das dem Schema einer relationalen Datenbank ähnelt. In diesem Wortschatz werden die Begriffe definiert, die benutzt werden, um die Relationen zwischen den Datenelementen herzustellen. Im Gegensatz zum relationalen Datenbankschema, ist das Vokabular für RDF über das ganze Web verteilt und wird von Menschen weltweit entwickelt. Neben den eigentlichen Begriffen und der dafür verwendeten URI, werden nützliche Zusatzinformationen auf der jeweiligen Webseite bereit gestellt. Prinzipiell kann jeder Mensch ein Vokabular definieren, es haben sich aber mittlerweile bekannte Initiativen wie SKOS[26],

[22] Vgl. Web Data Commons - RDFa, Microdata, and Microformats Data Sets - November 2013 http://webdatacommons.org/structureddata/2013-11/stats/stats.html (28.09.2016).
[23] Vgl. About microformats, http://microformats.org/wiki/about (05.09.2016).
[24] Vgl. HTML Microdata, https://www.w3.org/TR/2012/WD-microdata-20120329 (05.09.2016)
[25] Vgl. RDFa Core 1.1, https://www.w3.org/TR/rdfa-core/ (05.09.2016)
[26] Vgl. SKOS - Simple Knowledge Organization System, https://www.w3.org/2004/02/skos/ (05.09.2016)

FOAF[27] oder schema.org[28] etabliert. Für sehr spezifische Anwendungen kann aber ein eigenes Vokabular durchaus sinnvoll sein, so wurde auch für das InLOC Informationsmodell ein eigenes Vokabular definiert und auf der eigenen Seite zur Verfügung gestellt. Die am häufigsten benutzten, zentralen Begriffe werden als Kernvokabular (engl. *core vocabularies*) bezeichnet, die spezielleren, weniger verbreiteten als *authorititative vocabularies*.[29]

Eine Auflistung verschiedenster Vokabulare finden sich auf Linked Open Vocabularies[30]

## 2.2 Open Badges

Open Badges ist ein System digitaler Zertifikate, auch als Lern-Abzeichen bezeichnet, das von der Mozilla Foundation[31] und der MacArthur Foundation[32] gemeinsam entwickelt wurde. Open Badges können die Fähigkeiten und Leistungen eines Individuums abbilden und im Internet präsentieren. Schulen, Universitäten, Arbeitgeber und Anbieter informeller Bildung, benutzen Open Badges, um den Prozess lebenslangen Lernens zu erfassen und darzustellen. Abbildung 3 zeigt ein Beispiel eines Open Badges mit seinen Metadaten.

Abbildung 3: Beispiel eines Open Badges mit seinen Metadaten[33]

[27] Vgl. FOAF - Friend of a friend, http://www.foaf-project.org/ (05.09.2016)
[28] Vgl. Schema.org, https://schema.org/ (05.09.2016)
[29] [Woo14] Kapitel 2.3, RDF Vocabularies, S. 38
[30] Vgl. Linked Open Vocabularies, http://lov.okfn.org/dataset/lov/ (05.09.2016)
[31] Vgl. Mozilla Foundation, https://www.mozilla.org/en-US/foundation/ (10.08.2016)
[32] Vgl. MacArthur Foundation, https://www.macfound.org/ (10.08.2016)
[33] Quelle: http://badges.thinkoutloudclub.com/modules/what/know/

Es handelt sich bei Open Badges um einen offenen, technischen Standard, der von jeder Organisation oder Einzelperson genutzt werden kann, um Badges zu erstellen, zu vergeben und zu verifizieren. Badges können von verschiedensten Quellen zusammengetragen werden und in einem persönlichen digitalen "Rucksack"[34] (engl. *backpack*) gesammelt werden. Anschließend können die Fähigkeiten und Leistungen in Profilen sozialer Netzwerke oder auf persönlichen Webseiten angezeigt werden. Alle Open Badges können aufeinander aufbauen oder aus mehreren anderen Badges zusammengesetzt sein, auch wenn sie nicht von derselben Organisation ausgestellt wurden. Open Badges werden zwar als Bild dargestellt, die PNG-Datei enthält aber viele wichtige Metadaten, die auf den Aussteller, die Ausstellungskriterien und einen verifizierten Nachweis verlinken.[35]

Um eine zu stark voneinander abweichende Gestaltung der verschiedenen Badges zu vermeiden, gibt es spezielle Rahmenrichtlinien für das Design[36] sowie Programme die beim Entwerfen und Erzeugen von Badges helfen[37].

### 2.2.1 Open Badge Spezifikationen

Die von der *Badge Alliance* veröffentlichte Open Badges Spezifikation 1.1[38] erläutert die technische Aufteilung des Open Badge Standards auf drei verschiedene Badge Objekte. Die *Assertion* Klasse beschreibt eine individuelle Leistung oder Errungenschaft (engl. *accomplishment*), die generellen Merkmale der Leistung (engl. *achievement*) wird von der *BadgeClass* beschrieben und die Entität oder Organisation, die das Badge ausstellt, wird in der *Issuer* Klasse definiert. Die Abbildung 4 zeigt das Klassendiagramm der Badge Class, für den Prototypen sind hier speziell die markierten Felder *criteria* und *alignment* wichtig, da hier ein Verweis auf eine bzw. mehrere Kompetenzen erfolgen kann.

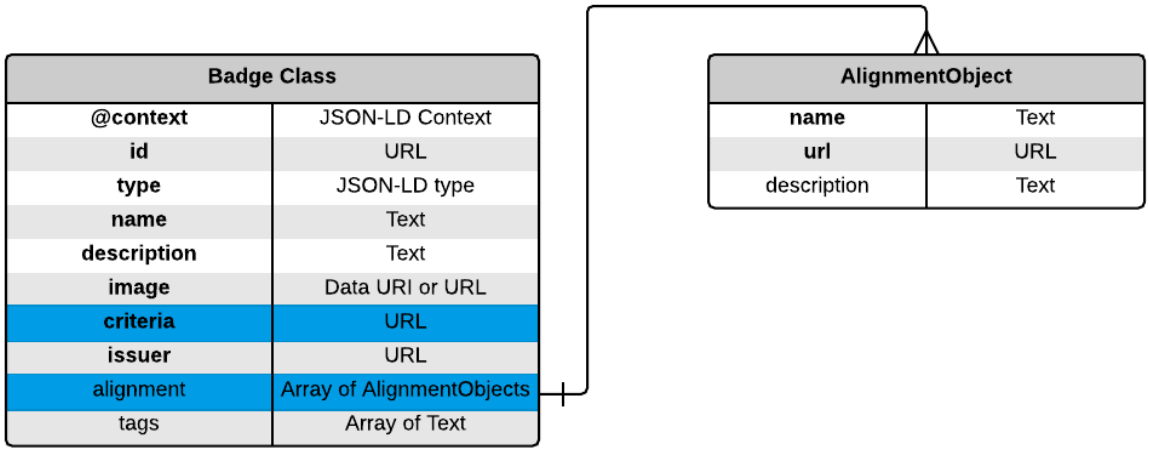


Abbildung 4 : Badge Class Spezifikation

[34] Vgl. Mozilla Backpack, https://backpack.openbadges.org/backpack (10.08.2016)
[35] Vgl. [Bad15]
[36] Vgl. Badge Creation Guidelines, http://badges.psu.edu/wp-content/uploads/sites/4454/2015/04/BadgeDesignGuidelines.pdf (20.08.2016)
[37] Vgl. Design and Issue Open Badges, https://www.openbadges.me/ (20.08.2016)
[38] Vgl. [Bad15]

Das *criteria* Attribut der BadgeClass repräsentiert die Kriterien für die Leistung, die notwendig ist, um sich das Badge zu verdienen. Wie in Abbildung… zu sehen, handelt es sich dabei um eine URL, die auf eine Webseite mit der Definition und erläuternden Informationen verweist. Das *alignment* Attribut enthält eine Liste von Objekten, die anzeigen sollen, an welchen Bildungsstandards sich das Badge ausrichtet. Wie in Abbildung 4 zu sehen, handelt es sich dabei um ein Array von *AlignmentObjects*, die wiederum aus Name, URL und Beschreibung zusammengesetzt sind. Im *url* Feld soll auf eine offizielle Beschreibung des Standards verlinkt werden, also Kompetenzspezifikationen aus einem offiziellen Rahmenwerk, wie zum Beispiel eComp oder e-CF handelt.

## 2.3 InLOC

Um ein Rahmenwerk für Kompetenzen abzubilden, benötigt man ein Modell, das bestimmte Strukturen und ein Vokabular vorgibt. In dem im Rahmen des Open Badges Network angefertigten Papers *Outcome 03-A2 - Competency Directory*[39] wird der InLOC Standard[40] als Informationsmodell für Kompetenz-Rahmenwerke vorgeschlagen. Dieser europäische Standard ist in der Lage, alle existierenden Kompetenz-Rahmenwerke in einer semantischen und Maschinen-lesbaren Form auszudrücken. Im folgenden sollen das InLOC Informationsmodell vorgestellt und dann an einem konkreten Beispiel erläutert werden, wie sich ein Kompetenz-Rahmenwerk in diesem Modell, darstellen lässt.

Neben InLOC gibt es noch weitere Informationsmodelle wie zum Beispiel den Standard *IEEE Standard for Learning Technology-Data Model for Reusable Competency Definitions*[41], der jedoch kommerziell ist und erworben werden muss.

### 2.3.1 Einführung

Der Begriff InLOC steht für *Integrating Learning Outcomes and Competences* und stellt ein Informationsmodell für Lernergebnisse (Learning Outcomes) und Kompetenzen (Competences), sogenannte *LOCs*, zur Verfügung. Ein Lernergebnis, als Resultat eines Lernprozesses, bestimmt, was bei einem Lernenden an Wissen, Verständnis und Fähigkeiten vorausgesetzt werden kann. Lernergebnisse können sowohl technisches Wissen und - technische Fähigkeiten beinhalten als auch "Soft Skills", wie soziale Fähigkeiten. Lernergebnisse werden oft in einem Lehrplan, einer Kursbeschreibung oder einem Qualifikations-Rahmenwerk definiert. Sie beziehen sich auf die

[39] Vgl. [Kon16] S.7 ff.
[40] InLOC - Integrating Learning Outcomes and Competences, http://www.cetis.org.uk/inloc/Home (12.08.2016)
[41] Vgl. IEEE Standard for Learning Technology-Data Model for Reusable Competency Definition, https://standards.ieee.org/findstds/standard/1484.20.1-2007.html (22.08.2016)

Struktur und Ziele einer Qualifikation.[42] Mit dem englischen Begriff *Competencies* werden Qualifikationen und spezifische Fähigkeiten und Kompetenzen beschrieben.[43]

InLOC definiert allgemeine Charakteristiken von LOCs und erlaubt damit die digitale Repräsentation dieser Strukturen und deren Austausch zwischen verschiedenen Systemen. Die meisten LOCs existieren als Teil einer Struktur oder eines Rahmenwerks, InLOC erlaubt es, die Beziehung zwischen den LOCs klar festzulegen. Das InLOC Modell benutzt ein eigenes, definiertes Vokabular. Jeder Begriff wird in einer eigenen Webseite definiert und hat damit eine eigene URI. Um die damit verbundene, benötigte Permanenz einer URL zu gewährleisten, wird jeweils eine Webadresse des Anbieters purl.org[44] verwendet, die dann auf die offizielle Seite des InLOC Standards (cetis.org[45]) weiterleitet.

### 2.3.1 Informationsmodell

Das InLOC Informationsmodell wird in [Cena13] S. 17 ff. näher beschrieben, dieses Kapitel bezieht sich darauf. Wie man in Abbildung 5 sehen kann besteht das InLOC Informationsmodell[46] aus der abstrakten Hauptklasse *LOC*, die alle möglichen Eigenschaften der beiden Unterklassen der LOC Struktur (*LOCstructure*) und der LOC Definition (*LOCdefinition*) enthält. Neben dem erforderlichen Identifizierer Attribut (*id*), spezifiziert InLOC weitere optionale Eigenschaften, wie Titel, Beschreibung, Sprache und Versionsnummer. LOC Struktur und LOC Definition stellen neben der LOC Assoziation Klasse (*LOCassociation*), die drei Hauptelemente des InLOC Informationsmodells dar. Durch LOC Assoziationen können die beiden anderen Klassen miteinander und untereinander, verknüpft werden. Im weiteren Verlauf der Arbeit benutze ich den Begriff "Struktur" als Synonym für LOCstructure, "Definition" als Synonym für LOCdefinition, "Assoziation" für die Klasse LOCassociation und den Begriff "LOC" als Zusammenfassung von LOCstructure und LOCdefinition.

[42] Vgl. [Cenb13],Kapitel 3 Background to InLOC - learning outcomes in learning education and training, S.9.
[43] Vgl. [Cenb13], Kapitel 4 Background to InLOC - competence in the world of employment, S.10.
[44] purl (Persistent Uniform Resource Locators) - https://purl.org/docs/index.html (08.08.2016)
[45] Vgl. InLOC Vokabular, http://www.cetis.org.uk/inloc/ (08.08.2016)
[46] Vgl. InLOC Classes and their constraints, [Cena13] S. 17 ff.

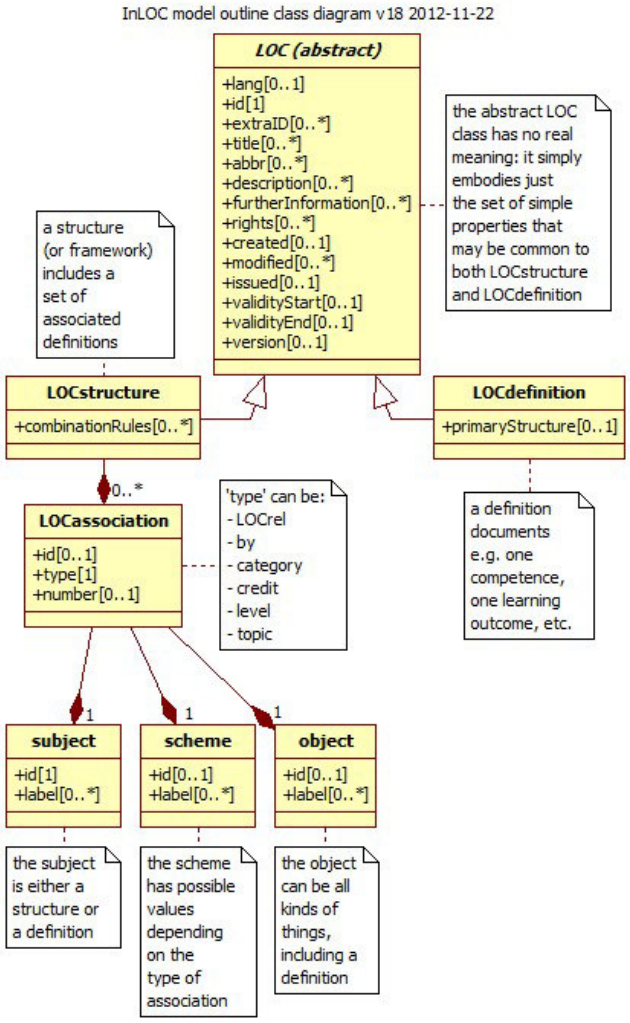


Abbildung 5: Klassendiagramm von LOCstructure, LOCdefinition und LOCassociation. Quelle: [Cena13] S.16

**Definition**

Eine Definition repräsentiert ein Konzept, das auf Wissen, Fähigkeit oder Kompetenz einer Person angewendet werden kann, es stellt also Lernergebnisse (engl. *Learning Outcomes*) und Arbeitskompetenzen (engl. *Competencies*) dar. Das Konzept, das die Definition repräsentiert, muss überprüfbar sein und in irgendeiner Form einer Bewertung unterzogen werden können. Eine Definition kann ein Konzept verschiedenster Granularität und Verallgemeinerung repräsentieren. Zur Festlegung eines Kontextes, kann über das Attribut der primären Struktur (*primary structure*), ein Verweis auf die zugehörigen Struktur hergestellt werden.

**Struktur**

Eine Struktur beschreibt im Gegensatz zur Definition kein einzelnes Konzept, das bewertet werden kann, sondern eine Struktur, die verschiedene ähnliche Definitionen, in einer Art Container zusammenfasst. Eine Struktur wird im Kontext der Kompetenz-Rahmenwerke, für die Zusammenfassung aller Unterelemente eingesetzt, repräsentiert also das Rahmenwerk ansich. Als zusätzliches Attribut kann die Struktur über beliebig viele Kombinationsregeln (*combinationRules*) verfügen, die Regeln für die Verknüpfung zu Definitionen festlegen.

Der Identifizierer (*id*) muss bei beiden Klassen global einzigartig sein, um ein LOC eindeutig vom anderen unterscheiden zu können und um so mit einen globalen Zugriff und eine mögliche Wiederverwendung zu garantieren. Titel und Namen sind nicht genug, da derselbe Titel in verschiedenen Kontexten, auf unterschiedliche LOCs verweisen könnte.

**Assoziation**

Eine Assoziation ist ein Mechanismus mit mehreren Funktionen. Es können damit strukturelle Beziehungen zwischen Struktur und Definition, sowie assoziative Beziehungen zwischen Definitionen abgebildet werden. Des Weiteren liefert sie eine Struktur, die es ermöglicht, Eigenschaften zu definieren, die mehr als eine Information enthalten, sogenannte zusammengesetzte Eigenschaften (*compound properties)*.

Eine Assoziation kann eine id und eine Nummer als Eigenschaft haben, und muss über Typ, Subjekt, Scheme und Objekt verfügen. Welcher Art Scheme und Objekt sind, wird von dem Typ der Assoziation bestimmt. Es gibt insgesamt 6 verschiedene Typen: *LOCrel, by, category, credit, level* und *topic*, wobei es sich bei allen außer *LOCrel* um zusammengesetzte Eigenschaften (*compound properties)* handelt. Zusammengesetzte Eigenschaften verknüpfen LOCs mit relevanten strukturierten Informationen.

**LOCrel**

Der Typ *LOCrel* modelliert eine Beziehung von Strukturen und Definitionen untereinander. Die allgemeinste Form *hasLOCpart*, zeigt an, dass die Struktur oder Definition mit einem anderen LOC verknüpft ist. Die spezielleren Formen dieses Typs zeigen an, dass es sich bei dem Kind-Element um ein Beispiel (*hasExample*), einen notwendigen Teil (*hasNecessaryPart*) oder einen optionalen Teil (*hasOptionalPart*) handelt. Für jeden dieser Typen gibt es eine Inverse, bei der es sich um eine Beziehung vom Kindelement zum Elternelement handelt. Die Inverse von *hasLOCpart* ist somit zum Beispiel *isLOCPartOf*.

Es gibt weiterhin Formen, die potentiell außerhalb der InLOC Struktur verlinken. Hiermit kann Bezug auf andere LOCs und Objekte außerhalb der eigenen InLOC Struktur genommen werden so kann zum Beispiel angezeigt werden, dass eine Definition Ähnlichkeit zu einer anderen Definition aufweist (*closeMatch*) oder dass beide Definition dasselbe Konzept repräsentieren (*exactMatch*). Mit *hasPreRequisite* kann eine Definition zur Voraussetzung einer anderen gemacht werden. Wenn eine neue Definition eine ältere ersetzen soll, kann die LOCrel Form *replaces* benutzt werden. Eine Zusammenfassung aller *LOCrel* Typen mit jeweiliger Erläuterung findet sich im Kapitel 7.6 *InLOC specified relationship* [Inla] auf Seite 41.

### 2.3.2 InLOC am konkreten Beispiel

Dieses Kapitel orientiert sich an der Veröffentlichung *InLOC - Part 2: Guidelines including the integration of Learning Outcomes and Competences into existing specifications* [Cenb13].
InLOC ist darauf ausgelegt eine Bandbreite von Frameworks und Strukturen repräsentieren zu können. Anhand eines Beispiels soll hier das Informationsmodell etwas anschaulicher erklärt werden. Dafür wurde das European e-Competence Framework e-CF[47] ausgewählt, welches Kompetenzen aus dem Arbeitsbereich Informations- und Kommunikations- technologie referenziert und dabei eine einheitliche Sprache zur Beschreibung von Kompetenzen, Fähigkeiten und Leistungsniveaus, verwendet. Dieses Rahmenwerk bietet sich zum einen an, da es verschiedene Eigenschaften enthält, die mit dem InLOC Standard modelliert werden können und zum anderen, weil es auch auf der offiziellen InLOC Website als Beispiel analysiert wird.

In der Version 3.0 des e-CF werden insgesamt 40 Kompetenzen referenziert und dabei in fünf Gebiete unterteilt: *Plan* (A), *Build* (B), *Run* (C), *Enable* (D), *Manage* (E).[48]

Die Struktur des Rahmenwerk setzt sich aus vier Dimensionen zusammen, in Abbildung 6 sieht man Dimension 1 bis 3. Die fünf Gebiete A bis E bilden die Dimension 1. Dimension 2 wird von den insgesamt 40 Kompetenzen gebildet und Dimension 3 beinhaltet die verschiedenen Stufen des Leistungsniveaus (*proficiency levels*). In Abbildung 6 nicht zu sehen ist Dimension 4, wo Beispiele für Fähigkeiten und Wissen angeführt werden, um die jeweilige Kompetenz zu veranschaulichen und überprüfbar zu machen.

| **Dimension 1**<br>5 e-CF areas<br>(A – E) | **Dimension 2**<br>40 e-Competences identified | **Dimension 3**<br>e-Competence proficiency levels<br>e-1 to e-5, related to EQF levels 3–8 | | | | |
|---|---|---|---|---|---|---|
| | | e-1 | e-2 | e-3 | e-4 | e-5 |
| A. PLAN | A.1. IS and Business Strategy Alignment | | | | | |
| | A.2. Service Level Management | | | | | |
| | A.3. Business Plan Development | | | | | |
| | A.4. Product/Service Planning | | | | | |
| | A.5. Architecture Design | | | | | |
| | A.6. Application Design | | | | | |
| | A.7. Technology Trend Monitoring | | | | | |
| | A.8. Sustainable Development | | | | | |
| | A.9. Innovating | | | | | |

Abbildung 6: Ausschnitt des e-CF, Gebiet A. Plan. Quelle: [Cen14] S. 19 Ausschnitt

Die Abbildung des e-CF auf den InLOC Standard soll hier speziell an der Kompetenz A.2 *Service Level Management* erklärt werden. Abbildung 6 zeigt die Details der Kompetenz A.2 in allen vier Dimension. Die verschiedenen InLOC Elemente werden in der Abbildung farblich unterschiedlich markiert. Die Struktur ist gelb umrahmt, Definitionen haben einen roten Rahmen und die Assoziationen sind durch orange, beschriftete Pfeile dargestellt.

[47] Vgl. European e-competence Framework, http://www.ecompetences.eu/, (letzter Abruf 05.09.2016)
[48] Vgl. [Cen14], S. 11 ff.

Jedes Kompetenz-Rahmenwerk ist in seiner Struktur verschieden. Die Unterteilung in vier Dimensionen ist zum Beispiel typisch für das e-CF, die Abgrenzung zwischen Struktur und Definition ist deshalb vom Anwendungsfall abhängig.

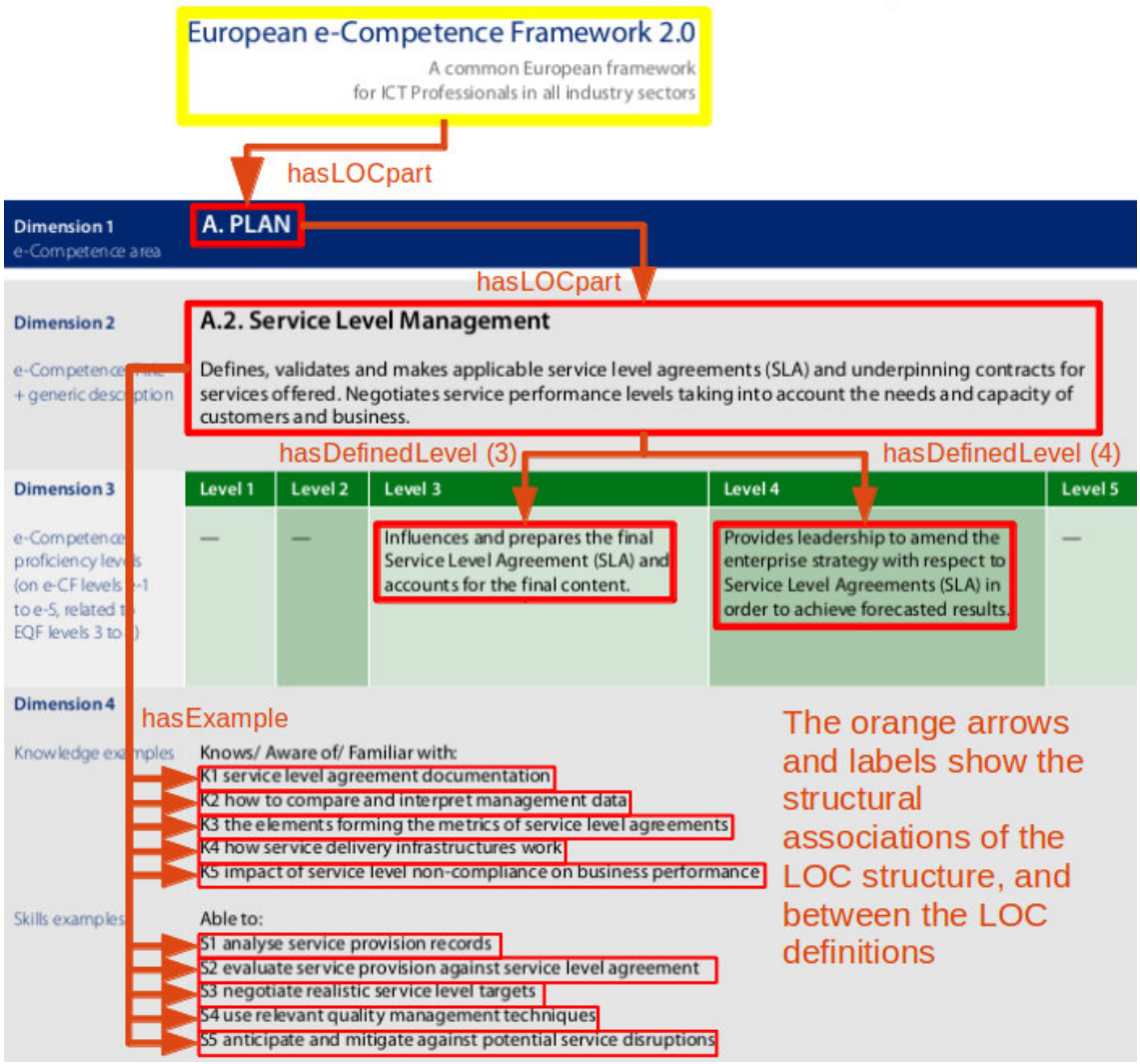


Abbildung 7: Beispielseite mit LOCstructure, LOCdefinition und LOCassociation,
Quelle: [Cenb13] S. 18, Figure 5.

In diesem Beispiel stellt das e-CF als Ganzes die einzige Struktur dar, sie ist in Abbildung 7 gelb umrahmt und fasst die Definitionen in einem Kontext zusammen. Neben Titel und Beschreibung hat das e-CF als Metadaten ein Veröffentlichungsdatum und eine Versionsnummer 2.0, die als Attribute gespeichert werden können. InLOC unterstützt Mehrsprachigkeit in Form von menschenlesbaren Strings, die in verschiedenen Sprachen angegeben werden können, wobei die Sprache durch einen Sprachencode mit zwei Buchstaben festgelegt wird. Die vorherrschende Sprache des gesamten Rahmenwerks kann auch in dem *language* Attribut der Struktur definiert werden.

Eine Definition hat im Gegensatz zur Struktur, keine strukturierende Aufgabe, sondern repräsentiert ein Konzept, das in irgendeiner Form überprüfbar sein muss. In Dimension 1 können die Fähigkeiten

eines Individuums in einem Gebiet durch die konkreten Kompetenzen dargestellt werden. Als Beispiele für die Fähigkeiten, die eine Kompetenz beschreiben, dienen zum einen die Leistungsniveaus der Dimension 3 und zum anderen die Wissens- und Fähigkeiten-Beispiele aus Dimension 4. Alle vier Dimensionen lassen sich also über eine Definition modellieren. Definitionen sollten immer einen Titel, eine Beschreibung oder beides haben. Da es keine eindeutige Trennung zwischen Titel und Beschreibung gibt, muss immer im Einzelfall entschieden werden, welches Attribut benutzt wird.

Jede Struktur und Definition muss einen einzigartigen, eindeutigen Identifizierer haben. Das e-CF verfügt über lokale, interne Identifizierer, die verschiedenen Gebiete werden über Großbuchstaben identifiziert. So hat das Gebiet *PLAN* z.B. den Bezeichner *A*, Kompetenzen als Unterteile haben zusätzlich eine Nummer, z.B. *A*.2 für *Service Level Management*. Um einen global eindeutigen Identifizierer zu erhalten, sollte eine Webadresse als Zusatz verwendet werden, z.B. die Webseite des e-CF:

**www.ecompetences.eu/uri/A2**

Auf dieser Seite sollten sich dann auch zusätzliche Information zu der jeweiligen Definition bzw. Struktur befinden.

Im Dokument ist die Verbindung der einzelnen Elemente, für den menschlichen Betrachter, offensichtlich, die Wissen-Beispiele gehören z.B. eindeutig zur Kompetenz, für eine flexible, elektronische Repräsentation, in Form einer Datenbank, müssen diese Assoziationen aber explizit gemacht werden. Die Assoziationen repräsentieren diese Beziehungen, hauptsächlich alle die Relationen zwischen Definitionen, einschließlich der Strukturen. Die Struktur eines Rahmenwerkes geht dabei vom Allgemeinen zum Speziellen. Beim e-CF z.B. ist die Kompetenz eine Unterteilung eines Gebietes, für diesen Ansatz ist die Assoziation *hasLOCPart* und die Inverse *isLOCPartOf* gedacht.

Das e-CF verknüpft seine fünf Gebiete mit einer *hasLOCpart* Beziehung und die Unterteilung dieser Gebiete in speziellere Kompetenzen erfolgt ebenfalls mit dieser Beziehung. In diesem Beispiel haben alle Gebiete die gleiche Wertigkeit, keines ist bedeutender als das andere. Mit der Assoziation *hasNecessaryPart* und *hasOptionalPart* könnte man aber eine solche Bewertung mit einbringen, beide Typen sind dabei Untertypen von *hasLOCPart*.

Die Beziehung zwischen der Kompetenz A.2 und den dazugehörigen Wissens- und Fähigkeiten-Beispielen ist etwas anders konzipiert. Die fünf Beispiele repräsentieren nicht alles, was man wissen muss, um über eine Kompetenz zu verfügen, sondern sind nur veranschaulichende Beispiele, die verändert und erweitert werden können. Die Assoziation zu den Wissens- und Fähigkeiten-Beispielen kann der Typ *hasExample* verwendet werden.

Die Beziehung zwischen einer Kompetenz und einem Leistungsniveau (engl. *proficiency level*) hat einen besonderen Charakter, da in dem Beispiel die Stufen 3 und 4 keine Komponenten von A.2 sind, sondern spezifisch definierte Stufen des generellen A.2 Konzepts. Für diesen Fall bietet der InLOC Standard den Typ hasDefinedLevel an. Die Nummer, die jede LOC Assoziation als optionale Eigenschaft besitzt, ist dabei entscheidend, denn sie erlaubt einen mathematischen Vergleich zwischen verschiedenen Stufen, bei der Annahme, dass höhere Stufen durch größere Zahlen repräsentiert werden. Die generischen Stufen (im e-CF e-1 bis e-5) beziehen sich auf das

Rahmenwerk als Ganzes und sind damit viel allgemeiner als Leistungsniveau-Stufen einer Kompetenz. Diese spezielleren Stufen der Kompetenz sind im Prinzip Beispiele der generischen Stufen und können somit am besten durch eine Assoziation vom Typ *hasExample* repräsentiert werden.

Für die zusammengesetzten Eigenschaften (engl. *compound properties*), wie zum Beispiel *category* oder *topic,* werden ebenfalls Assoziationen verwendet. Ein Thema (*topic*) wird so nicht durch einen einfachen String dargestellt, sondern als Begriff aus einem speziellen Katalog von Schemata oder Vokabeln. In der Assoziation repräsentiert die Klasse *scheme* dabei den Themenkatalog und die Klasse *object* den Begriff in diesem Katalog.

## 2.4 Relevante ähnliche Arbeiten

Im Rahmen der Recherche zu verwandten Arbeiten wurden einige Projekte gefunden, die Wissensverzeichnisse implementiert haben. Zwei Beispiele dafür sind *ariadne*[49] und *dspace*[50] In diesen Wissensverzeichnissen kann Wissen gespeichert, ausgetauscht und eingebunden werden, jedoch stellt keines dieser Projekte ein Verzeichnis für Kompetenzendefinitionen dar.

Es wurden jedoch zwei Arbeiten identifiziert, in deren Rahmen ebenfalls ein digitales Verzeichnis für Zertifikate bzw. Kompetenzen implementiert wurde. Nach bestem Wissen des Autors bestehen keine weiteren ähnlichen Lösungen, dennoch kann nicht ganz ausgeschlossen werden, dass weitere, nicht gefundene, Lösungen, existieren. Diese zwei Arbeiten sollen im Folgenden kurz vorgestellt werden.

**Open Badges Verzeichnis**[51]
2015 hat die Open Badge Alliance eine Initiative gestartet, um existierende Badges in einem Verzeichnis zu sammeln. Herausgeber von Badges können ihre eigene Seite registrieren lassen um sie zu indexieren und dem Verzeichnis hinzuzufügen. Das Verzeichnis stellt eine API zur Verfügung, um nach Kriterien zu suchen. Dabei können auch Kategorien wie *kürzlich indexiert* abgefragt werden, oder alle Badges einer bestimmten Domain abgerufen werden.

Das Verzeichnis kann beim Vergleich von Informationen im alignment- oder criteria-Feld der BadgeClass helfen, funktioniert jedoch nicht auf Basis semantischer Metadaten. Das Verzeichnis ist für Open Badges konzipiert und macht einen Vergleich von Kompetenzen nur indirekt möglich.

**Compbase**
Compbase[52] ist ein Projekt der Universität Potsdam und stellt ein Verzeichnis für Kompetenzen auf Basis einer Graphdatenbank, dar. Im Gegensatz zu dem in dieser Bachelorarbeit erstellten Prototypen, benutzt *Compbase* nicht den InLOC Standard und basiert auch nicht auf Linked Data. Die Kompetenzdefinitionen eines Rahmenwerks müssen manuell eingetragen werden und werden nicht automatisch über ein HTML-Dokument mit Micro-Data oder ein JSON-LD Dokument eingelesen.

[49] Vgl. ariadne, http://www.ariadne-eu.org (15.09.2016)
[50] Vgl. dspace, http://www.dspace.org/ (15.09.2016)
[51] Vgl. Openbadges-directory, https://github.com/badgealliance/openbadges-directory (08.08.2016)
[52] Vgl. COMPBASE, https://github.com/uzuzjmd/COMPBASE (08.08.2016)

# 3. Anforderungsanalyse

In diesem Kapitel sollen zunächst die Erkenntnisse aus Kapitel 2 zusammengefasst und analysiert werden, um daraus ein grundlegendes Konzept für einen Prototypen zu entwickeln. Danach werden die konkreten Anwendungsfälle beschrieben und in den Abschnitten 3.4 und 3.5 die daraus resultierenden Anforderungen an eine Implementierung definiert.

## 3.1 Grundlegendes Konzept für einen Prototypen

In Kapitel 2.2 wurde der Open Badge Standard näher untersucht und es wurde gezeigt, dass ein Open Badge über die Felder *alignment* und *criteria* auf ein, oder mehrere Kompetenzdefinitionen verlinkt werden kann. Im bisherigen Ansatz verlinken die Badges auf eine Webseite auf der eine bestimmte Kompetenz definiert ist oder eine Definition, zum Beispiel als PDF-Datei, abrufbar ist. Der Nachteil dabei ist, dass die Informationen nur als Text vorliegen. Dessen Struktur und die Bedeutung der verschiedenen Teile, wie zum Beispiel des Titels oder der Beschreibung, kann zwar der Mensch interpretieren, ein Programm allerdings nicht, ohne weiteres. Könnte ein Programm jedoch die verschiedenen Bestandteile einer Kompetenz oder eines Kompetenz-Rahmenwerkes analysieren und deren Bedeutung interpretieren, so wäre es zum Beispiel möglich, verschiedene Kompetenzen miteinander zu vergleichen oder andere Rückschlüsse auf die Eigenschaften der Kompetenzen zu ziehen.

Die in Kapitel 2.1 vorgestellten Konzepte des *semantischen Webs* und *Linked Data* ermöglichen genau das. Vorrausetzung für eine Anwendung dieser Konzepte ist jedoch, dass die Daten, in diesem Fall die Kompetenzdefinitionen, mit semantischen Informationen in Form des RDF Datenmodells vorliegen. Für die Einbindung dieser semantischen Informationen kann eins der verschiedenen RDF-konformen Formate, wie RDFa oder JSON-LD benutzt werden. Die semantische Annotation hat dabei den Vorteil, dass die Informationen gleichzeitig menschen- und maschinenlesbar sind.

Um die verschiedenen Teile einer Kompetenz bzw. eines Kompetenz-Rahmenwerkes zu unterscheiden und zu definieren, braucht es ein Modell. In Kapitel 2.3 wurde dafür das InLOC Modell vorgeschlagen und analysiert. Anhand des Beispiels e-CF wurde gezeigt, dass es möglich ist, ein Rahmenwerk in diesem Informationsmodell in eine strukturierte, semantische Form zu bringen. Gleichzeitig stellt InLOC ein Vokabular zur Verfügung, mit dem verlinkte, semantische Daten in eine Webseite oder ein Dokument eingebunden werden können.

Angenommen, die Kompetenzdefinitionen sämtlicher existierender Kompetenz-Rahmenwerke würden in Form des InLOC Modells und in Form strukturierter, semantischer Daten vorliegen: um diese Kompetenzdefinitionen in ein Open Badge einzubinden, würde noch eine Möglichkeit fehlen, nach diesen Kompetenzen zu suchen und ihre URI zu erhalten. Diese Aufgabe soll ein Verzeichnis übernehmen, dass als Prototyp im Rahmen dieser Bachelorarbeit entstehen soll.

Das Verzeichnis soll Kompetenzdefinitionen und Kompetenz-Rahmenwerke, die als semantische, verlinkte Daten vorliegen, einlesen können und einem Benutzer dann die Möglichkeit geben, in diesem Verzeichnis nach einer Kompetenz zu suchen und die URI für eine bestimmte Kompetenzdefinition zu erhalten.

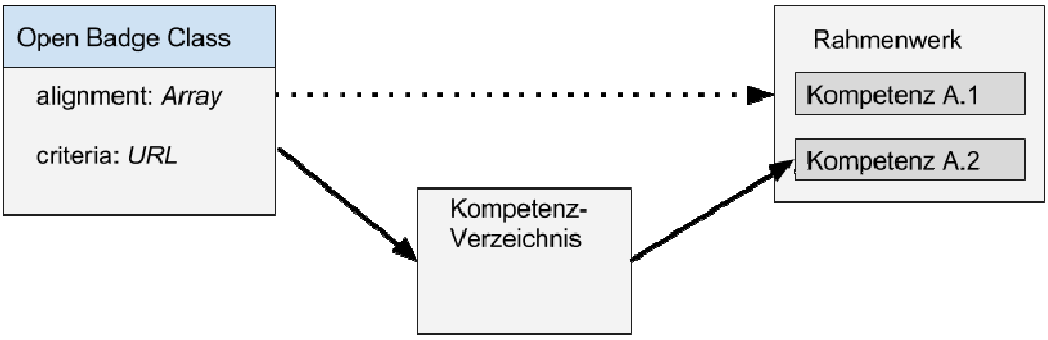


Abbildung 8: Das Kompetenz-Verzeichnis verbindet Open Badge und Kompetenz- Rahmenwerk

Da eine URI gleichzeitig auch eine URL ist, kann sie als Wert in dem *alignment* oder *criteria* Feld des Open Badge angegeben werden und damit eine Kompetenz eindeutig identifizieren.
Die Abbildung 8 zeigt, wie das angesprochene Kompetenz-Verzeichnis die Lücke zwischen dem Open Badge und den Kompetenzdefinitionen schließen kann.

In den nachfolgenden Kapiteln 3.2 bis 3.4 sollen die genauen Anforderungen, die der Prototyp erfüllen soll definiert werden.

## 3.2 Voraussetzungen

Um die Funktion des Prototypen zu garantieren, muss das einzulesende Kompetenz-Rahmenwerk in einem bestimmten Format vorliegen. Die einzelnen Komponenten müssen der Definition des InLOC Informationsmodells entsprechen und das definierte InLOC Vokabular benutzen. Die Daten müssen im RDFa Format vorliegen. Zum Testen des Prototyps wurden Teile des bereits vorgestellten Kompetenz-Rahmenwerks e-CF in das InLOC Modell übertragen und in Form einer Webseite mit RDFa formatierten Daten gespeichert. Dieses Dokument befindet sich im Anhang[53]

Für die Festlegung der URIs der einzelnen InLOC Elemente eines Rahmenwerkes gibt es noch keine klare Vorgehensweise. Als möglicher Ansatz wurde beim Beispiel des e-CF eine Erweiterung der URL gewählt. Über den zusätzlichen Pfad */uri* kann so auf die einzelnen Elemente verwiesen werden. Eine URI der Kompetenz A.2 könnte dann wie folgt angegeben werden:

*http://www.ecompetences.eu/uri/A2*

Diese URL dient dabei momentan nur als Identifikator, ohne dass sich dahinter eine Webseite verbirgt. Bei einer tatsächlichen Umsetzung dieses Ansatzes, sollte die URL aber auf eine Webseite, mit den Informationen zu dem Element verlinken.

Ein anderer Ansatz wäre die Verwendung von Ankerpunkten, die dann auf einen Abschnitt innerhalb einer Webseite verlinken. Die URI würde dann wie folgt aussehen:

*http://www.ecompetences.eu/uri#A2*

Nachteil dieses Ansatzes ist, dass sich alle Informationen tatsächlich in einem Dokument befinden müssten und nicht auf verschiedene Dokumente auf dem Server verteilt wären.

[53] CD-Pfad: *RDFa/e-CF.html*

## 3.3 Anwendungsfälle

Die in Kapitel 3.1 angesprochenen Aufgaben des Kompetenz-Verzeichnisses sollen an dieser Stelle durch konkrete Anwendungsfälle definiert werden. Abbildung 9 zeigt die drei Anwendungsfälle, die das Kompetenzverzeichnis abdecken soll:

1. Der Benutzer will eine Kompetenz oder ein Kompetenz-Rahmenwerk hinzufügen und übermittelt dafür eine URL.
2. Der Benutzer will nach einer Kompetenz suchen.
3. Der Benutzer will die URI einer bestimmten Kompetenz erhalten.

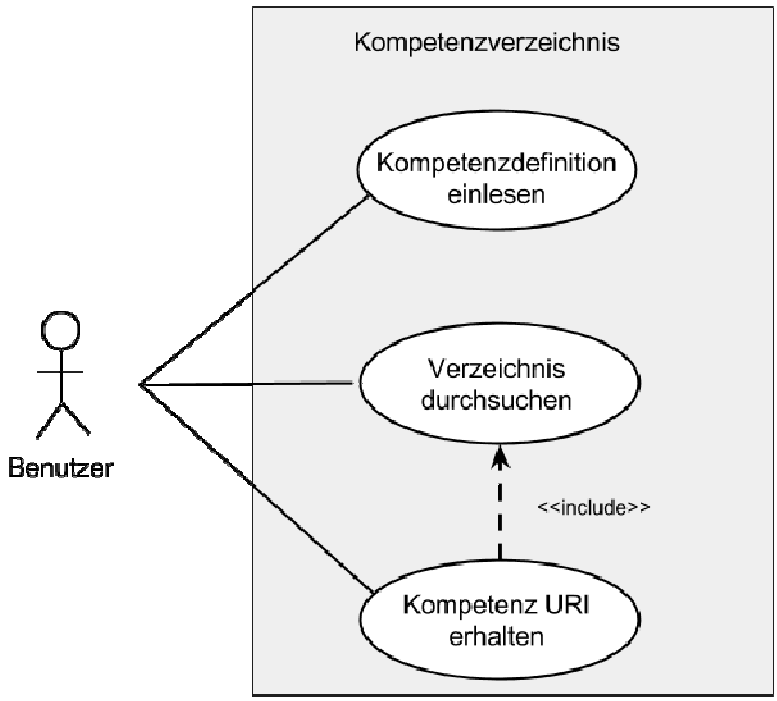


Abbildung 9: Anwendungsfalldiagramm

In der Abbildung 9 gibt es nur einen Akteur, den Benutzer. Bei einer tatsächlichen Umsetzung sollte es aber eine Authentifizierung geben, da nicht jeder Benutzer dazu autorisiert sein soll, neue Einträge anzulegen. In diesem Fall gäbe es zwei Akteure, den angemeldeten Benutzer und den nicht angemeldeten Benutzer. Für den Prototypen wird eine Authentifizierung vernachlässigt.

## 3.4 Funktionale Anforderungen

Aus den grundlegenden Gedanken der Applikation (Abschnitt 3.1) und den Anwendungsfällen leiten sich die in Tabelle 1 definierten funktionalen Anforderungen ab.

| | |
|---|---|
| **F.1** | Um neue Einträge in die Datenbank einzufügen, sollen semantische, strukturierte Daten aus einer Webseite extrahiert werden |
| **F.2** | Der Benutzer kann die Einträge, die eine Kompetenz definieren anhand von Suchbegriffen durchsuchen |
| **F.3** | Die Suchergebnisse werden in Listenform angezeigt |
| **F.4** | Der Benutzer erhält eine Detail-Ansicht der ausgewählten Kompetenz |
| **F.5** | Der Benutzer kann sich die URI, die eine Kompetenz identifiziert, anzeigen lassen |

Tabelle 1: Funktionale Anforderungen

## 3.5 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen sind wie in Tabelle 2 dargestellt definiert.

| | |
|---|---|
| **NF.1** | Die Daten sollen so gespeichert werden, dass deren semantische Informationen nicht verloren gehen |
| **NF.2** | Es soll eine API zur Verfügung gestellt werden, um dezentral von verschiedenen Clients auf das Verzeichnis zugreifen zu können |
| **NF.3** | Das Verzeichnis soll zentral alle Daten zusammenfassen |
| **NF.4** | Das System extrahiert Daten im RDFa Format, die dem InLOC Vokabular entsprechen, aus einer Webseite |
| **NF.5** | Der Benutzer soll auf Fehler in der Eingabe oder des Systems hingewiesen werden |

Tabelle 2: Nichtfunktionale Anforderungen

# 4. Entwurf

Aus den Anforderungen die in Kapitel 3 definiert wurden, soll nun ein Entwurf für einen Prototypen entwickelt werden. Zunächst soll dabei die Systemarchitektur betrachtet werden.

## 4.1 Systemarchitektur

Aus den nichtfunktionalen Anforderungen aus Kapitel 3.5 geht hervor, dass die Daten zentral in einem Verzeichnis gespeichert werden (NF.3) und außerdem eine Schnittstelle zur Verfügung gestellt werden soll, damit verschiedene Clients dezentral auf das Verzeichnis zugreifen können (NF.2). Diese Anforderungen entsprechen einer Client-Server-Architektur. Das Backend wird dabei von einem Server, der auf eine Datenbank zugreift, gebildet. Client und Server kommunizieren über eine Internetverbindung miteinander und tauschen Daten aus. Der Server behandelt alle Anfragen des Clients und liefert Daten zurück. Er fügt der Datenbank neue Einträge hinzu und liefert, durch Suchbegriffe gefilterte, Datenbankeinträge an den Client zurück.

### 4.2.1 Kommunikation zwischen Client-Applikation und Server

Die Anforderung NF.2 definiert eine API, die dezentral von verschiedenen Clients aus abgerufen werden soll. Dabei kann es sich potentiell um verschiedene Arten von Clients handeln Vorstellbar ist also nicht nur eine Webseite in einem Browser sondern zum Beispiel auch eine mobile Applikation auf einem Smartphone. Die Schnittstelle sollte also möglichst lose an den Server gekoppelt sein. Für diesen Anwendungsfall bietet sich ein Webservice an, insbesondere eine REST Schnittstelle.[54]

[54] Vgl. Representational State Transfer, https://de.wikipedia.org/wiki/Representational_State_Transfer

**REST API Endpunkte**

Die Erzeugung eines neuen Eintrags in dem Verzeichnis erfolgt durch das Senden einer URL an den Server, dafür wird ein Endpunkt, mit POST-Request benötigt.

Da es, zumindest in dieser Version des Prototypen, nicht vorgesehen ist manuell Einträge des Verzeichnisses zu erstellen, bearbeiten und zu löschen, wird nur ein Endpunkt für den Abruf aller Kompetenzen und ein Endpunkt für den Abruf einer einzelnen Kompetenz benötigt. Zusätzlich sollte es zu Testzwecken noch eine Methode zum Löschen des gesamten Verzeichnisses geben, da beim Eintragen nur Identifikatoren akzeptiert werden, die sich noch nicht in der Datenbank befinden.

Um im Verzeichnis zu Suchen, wird ein Endpunkt mit POST-Request bereitgestellt, dabei wird der Suchbegriff an den Server übermittelt.

| **HTTP-Methode** | **Route** | **Aufgabe** |
|---|---|---|
| GET | /competences | Liste aller Kompetenzen |
| POST | /competence | Liefert Kompetenz über eine ID |
| DELETE | /competences | Löschen aller Datenbankeinträge |
| POST | /search | Suchen nach Kompetenzen über Suchbegriff |
| POST | /submit | Übermitteln einer URL um RDFa Daten einzutragen |

Tabelle 3: Liste aller API Endpunkte

## 4.2 Datenhaltung

Da die Wahl der Datenbank eine Entscheidung über Programmiersprache bzw. Framework beeinflusst, soll vor den Überlegungen zu Server und Client zunächst die Struktur der Daten analysiert werden.

InLOC stellt verschiedene Möglichkeiten zur Verfügung, um das Informationsmodell zu implementieren. Die Bindung des Modells an ein konkretes Format beinhalten die Abbildung der Komponenten des Modells auf die Komponenten der ausgewählten Technologie. Das InLOC Team empfiehlt die Formate XML, RDF und JSON, bzw. JSON-LD. Für diese Formate ist eine Bindung zum InLOC Standard definiert. RDFa wird außerdem als wahrscheinlichstes Format für die Zukunft angesprochen.[55]

Nach Anforderung… sollen die Daten so gespeichert werden, dass ihre semantische Information erhalten bleibt. Anforderung .. legt außerdem fest, dass die Daten in einem RDF konformen Format eingelesen werden und auch in solch einem Format an den Benutzer übergeben werden sollen. Es liegt daher nahe, ein Format zu benutzen, dass geeignet ist, über eine REST Schnittstelle übertragen zu werden als auch in einer Datenbank gespeichert zu werden. JSON-LD erfüllt diese Anforderungen. Wie in Kapitel 2.1.7 aufgezeigt wurde, stellt ein JSON-LD Dokument gleichzeitig ein valides RDF Modell, als auch ein JSON-Objekt dar.

In der Online-Dokumentation von InLOC findet man Beispiele und Erklärungen, wie das InLOC-Modell als JSON-LD Objekt abgebildet werden kann. Wie bereits in Kapitel 2.1.7 erwähnt werden die semantischen Daten, welche die einzelnen Elemente definieren, in einem sogenannten Kontext angegeben. Für die Definition eines Attributs ist es dann nicht nötig jeweils die URI anzugeben, sondern es kann ein kurzer String als Schlüssel verwendet werden. Die verschiedenen URIs werden dann im Kontext festgelegt. InLOC stellt den Kontext als abrufbare Datei[56] zur Verfügung. In einer Implementierung könnte der Kontext dann automatisch abgerufen werden, um die semantischen Daten zu validieren.

Für die Speicherung von JSON-Objekten bietet sich eine Dokumentenbasierte Datenbank wie MongoDB an. In den Folien Gregg Kellog werden konkrete Vorschläge für eine Einbindung von JSON-LD in MongoDB gemacht (vgl. [Kel16]). Diese Art der Datenbank speichert die Einträge nicht, wie eine relationale Datenbank in Form von definierten Tabellen, sondern als Schemafreie Dokumente.

Um die LOC-Klasse in Form eines Datenbankmodells abzubilden bietet sich die Verwendung eines Hilfsprogramm zur Objektmodellierung an. Mit Mongoose steht ein solches Programm für die MongoDB Datenbank zur Verfügung. Es lassen sich damit Datenbankschemata definieren, indem alle gültigen Schlüssel mit ihrem jeweiligen Datentyp definiert aufgelistet werden. Das Schema dient dann als Grundlage für ein Modell, das sämtliche Methoden zur Abfrage von MongoDB bereitstellt. Es können zudem auch eigene Methoden definiert werden.[57]

---

[55] Vgl. InLOC Technical Bindings, http://www.cetis.org.uk/inloc/Bindings (13.09.2016)
[56] InLOC JSON-LD Kontext, http://purl.org/net/inloc/attachments/InLOC_context.jsonld, (13.09.2016)
[57] Vgl. Mongoose, http://mongoosejs.com/, (05.09.2016)

## 4.3 Server

**Architektur des Server-Backends**
Die Architektur des Server-Backends gliedert sich in drei Schichten (vgl. Abbildung 10). Die erste Schicht bildet die Zugriffsschicht auf die Anwendung und wird durch den REST-Service repräsentiert. Dieser nimmt die Anfragen des Clients an und wandelt die Eingabe, sowie die Antworten der darunterliegenden Services in den erwarteten Datentyp um. Die direkt darunterliegende Schicht beinhaltet die Logik, die notwendig ist, um die Anfragen der darüberliegenden Schicht zu behandeln und Daten in die Datenbank zu schreiben. Der Parser soll eine übergebene URL verarbeiten und extrahierte Daten in die Datenbank schreiben.

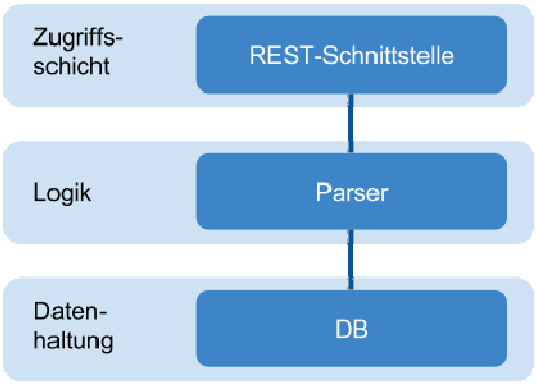


Abbildung 10: Die drei Schichten des Servers

Da im vorangegangenen Kapitel zur Datenhaltung (4.2) entschieden wurde, die Dokumentenorientierte Datenbank MongoDB zu verwenden, bietet es sich an, auch auf Server-Seite JavaScript zu verwenden. Die serverseitige Plattform *Node.js* bietet sich besonders dafür an einen Webserver auf Basis von JavaScript umzusetzen.[58]

**Parser**
Um die Daten, die eine Kompetenz beschreiben, in die Datenbank eintragen zu können, müssen diese zunächst in der Webseite gefunden und extrahiert werden. Diese Aufgabe soll ein Parser übernehmen. Die Daten sollen dabei in einem RDF-konformen Format vorliegen. Als Parameter soll eine URL oder ein Webdokument an den Parser übergeben werden, zurückgeliefert werden sollen die extrahierten Daten in JSON-LD. Der Parser soll dabei auf die Verwendung des richtigen Vokabulars achten, gültig ist dabei nur das Vokabular des InLOC Modells. Es ist davon auszugehen, dass es bereits Programme gibt, die für den Prototypen verwendet werden können.

[58] Vgl. Node.js, https://de.wikipedia.org/wiki/Node.js

## 4.4 Client

Wie in Abschnitt 4.2.1 dargestellt, soll die Kommunikation zwischen Server und Client über eine REST Schnittstelle erfolgen. Der Client stellt HTTP Anfragen und übergibt dabei gegebenenfalls Daten. Der Server liefert JSON-LD Objekte zurück, die vom Client entsprechend dargestellt werden.

**Struktur des User Interfaces**
Die in Abschnitt 3.2 drei Anwendungsfälle sind so unterschiedlich, dass es sich anbietet, dafür jeweils unterschiedliche Seiten oder Fenster zu benutzen. Es ergeben sich daraus folgende drei Seiten:

1. Eingabe und Übermittlung einer URL
2. Suche mit Anzeige der Ergebnisse
3. Detailansicht mit der Möglichkeit die URI der Kompetenz zu erhalten

Daneben sollte noch eine weitere Seite existieren, auf der das Projekt kurz vorgestellt wird und dem Benutzer die grundlegenden Funktionen der Anwendung erklärt werden. Dies kann auf einer Startseite erfolgen.

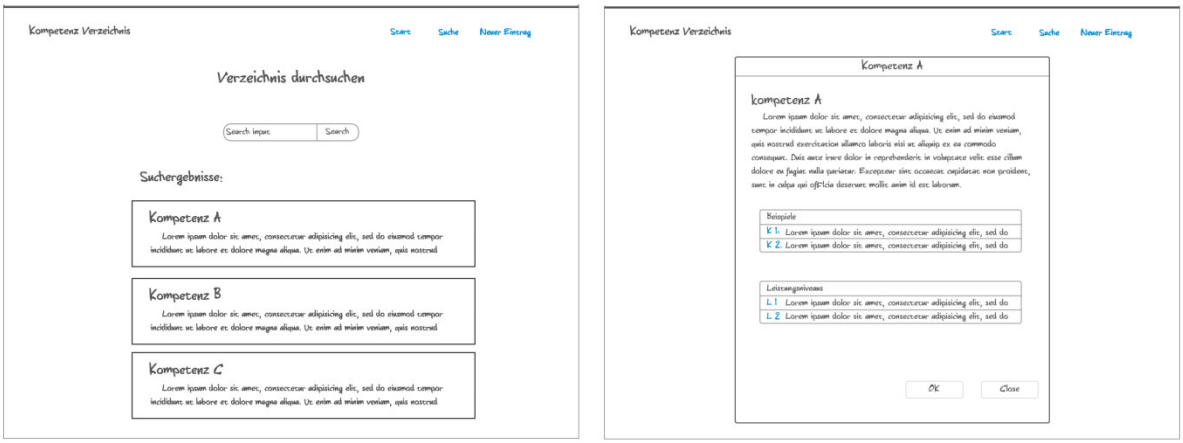


Abbildung 11: Visueller Prototyp der Client Applikation

Die Abbildung 11 zeigt einen visuellen Prototypen der Seite “Suche”. Hierbei soll eine Möglichkeit gezeigt werden, wie die Resultate einer Suchanfrage angezeigt werden können. Der Benutzer erhält die Suchergebnisse in Form einer Liste, es wird dabei nur der Titel und die Beschreibung eines Ergebnisses angezeigt. Weitere Informationen werden erst bei der Auswahl eines der Ergebnisse, in einem sich öffnenden Fenster angezeigt. In der Detailansicht werden dann auch die verschiedenen Assoziationen, wie etwa die verschiedenen Leistungsniveau-Stufen angezeigt.

Die Seite von der aus ein Benutzer eine URL übermitteln kann enthält nur ein Eingabefeld mit Schaltfläche. Der Benutzer erhält nach der Übermittlung der URL eine Rückmeldung darüber, ob die Daten erfolgreich in die Datenbank eingetragen wurden, oder ob ein Fehler aufgetreten ist.

# 5. Umsetzung

Auf Basis des in Kapitel 4 vorgestellten Enwurfs wurde ein funktionaler Prototyp bestehend aus einer Client-Applikation und einer Server-Applikation entwickelt. Der Prototyp besitzt den vollen, in Kapitel 3 definierten Funktionsumfang. In diesem Kapitel soll nun näher auf die Details der Implementierung eingegangen und besondere Teile des Codes sollen hervorgehoben werden. Bereits im Entwurf wurde festgelegt, dass für die Implementierung die Plattform *Node.js* verwendet werden soll in Kombination mit dem serverseitigen Web Application Framework *Express.js*. Als Datenbank wird die dokumentenbasierte Datenbank *MongoDB* benutzt.

## 5.1 Implementierung des Server-Backends

In diesem Kapitel sollen die verschiedenen Klassen, aus denen sich die Server-Applikation zusammensetzt vorgestellt werden.

**app** (*app.js*)
Die Datei ist der Startpunkt der Anwendung. Das Framework *Express.js* wird hier initialisiert und die Datenbank über *Mongoose* gestartet. Die Routen, befinden sich in einer separaten Datei (routes.js) werden hier aber als *Middleware* eingebunden, damit die Http-Anfragen an die Controller weitergeleitet werden können. Alle Anfragen, die keiner der definierten Routen entsprechen, werden am Ende abgefangen und mit einer Fehlermeldung beantwortet. Zum Schluß wird der Server gestartet. Der Server liefert auch die Client-Applikation aus, die sich im *public* Ordner befindet.

**routes** (*routes/routes.js)*
In dieser Datei wird der in *Node Express* integrierte Router verwendet um Routen zu definieren und die Http-Anfragen an die Methoden der beiden Controller weiterzuleiten.

**locController** (*controllers/locController.js*)
Dieser Controller stellt die Datenbankabfragen und benutzt dafür das Modell. Die verschiedenen Methoden entsprechen den Routen. Es werden einzelne LOCs und eine Liste mit allen LOCs abgerufen. Auch die Methode zur Datenbanksuche befindet sich hier. In Listing … sieht man, das mit einem regulären Ausdruck in den Feldern *description, title* und *abb* (Abkürzung) gesucht wird. Zu Testzwecken enthält der LOCController außerdem noch eine Methode zum Löschen aller Einträge der Datenbank.

```
search: function (req, res, next) {
       var term = req.body.query;
       console.log("search term: "+term);

       var re = new RegExp(term, 'i');

       LOC.find().or([
                {'description.en': { $regex: re}},
                {'title.en': {$regex: re}},
                {'abbr.en': {$regex: re}}
             ]).exec(function(err, locs){
                  if(err) return next(err);
                  res.status(200).send(locs);
             }
       );
}
```

Listing 2: Suche im locController

**parseController** (*controllers/parseController.js*)
Wie in Listing 3 zu sehen, wird in der *importData* Funktion eine URL aus dem Request- Objekt entnommen und an den Parser übergeben. Der Parser ruft die URL auf und extrahiert die Daten aus der Webseite. Wenn die Daten dem InLOC-Modell entsprechen, werden sie in die Datenbank eingefügt. Für das gleichzeitige Einfügen mehrerer Daten stellt *Mongoose* die komfortable Methode *insertMany* zur Verfügung. Die zweite Funktion in der Datei liest, zu Testzwecken, eine lokale Datei mit RDFa Daten ein und fügt sie in die Datenbank ein.

```
importData: function (req, res, next) {

       var url = req.params.url;
       console.log("url "+url);

       parser.parse(url, function(err, obj){
          if(err){
             var error = new Error('No Competencies found under that URL.');
             error.status = 404;
       } return next(error);

       LOC.insertMany(obj, function(err, docs){
          if(err) return next(new Error('The Competencies already exist in the
repository.'));

          res.status(200).send("Saved Elements");
       });
      });
  },
```

Listing 3: Die importData Funktion des parseControllers

**parser** (*lib/parser/parser.js*)
Der Parser benutzt die quelloffenen Bibliotheken *jsonld.js*[59] und *jsonld-rdfa-parser*[60] um semantische Daten, in Form von RDFa, aus den abgerufenen Webseiten zu extrahieren. *Jsonld.js* ist ein Prozessor, der die JSON-LD Spezifikation implementiert. Damit kann zum Beispiel ein JSON-Objekt um einen Kontext erweitert werden oder die Schlüssel können durch ihre im Kontext definierten URIs ersetzt werden. Die Umwandlung funktioniert dabei in beide Richtungen. Die Bibliothek erlaubt weiterhin die Einbindung eines externen RDFa Parsers. Dafür wurde der *jsonld-rdfa-parser* verwendet, da er speziell auf *jsonld.js* aufbaut. Der Parser nimmt URLs und HTML-Strings entgegen und extrahiert die gefundenen RDFa Daten. Diese werden anschließend automatisch in das JSON-LD Format umgewandelt.

Das parser Modul besitzt die Methoden *parse* und *parseUrl*, damit auch lokale Dateien eingebunden werden können, zur Ausführung von Tests (vgl. Listing 4). Die *parseUrl* Funktion benutzt das interne *Node.js* Modul *request*, mit dem Webseiten abgerufen werden können. Das Ergebnis wird an die *parse* Methode weitergeleitet, wo der RDFa Parser die Daten extrahiert. Um zu validieren, dass die Daten dem InLOC Vokabular entsprechen, wird der InLOC Kontext auf die Daten angewandt, was im Methodenaufruf *compact* stattfindet.

```
exports.parseUrl = function (url, callback) {
  callback = callback || function() {};

  request(url, function(err, res, body) {
       if (err) return callback(err);

       parse(body, callback);
  });
};

exports.parse = function (doc, callback){
  callback = callback || function() {};
  //extract RDFa data
  jsonld.fromRDF(doc, {format: 'text/html'}, function(err, data) {
       if (err) return callback(err);

       //compact with JSON-LD context
       jsonld.compact(data, context, function(err, compacted) {
       if (err) return callback(err);

       //remove annotation for DB
       convertToJSON(compacted, function(err, obj){
       if (err) return callback(err);

       return callback(null, obj);
       });
       });
  });
};
```

Listing 4: Die parse-Funktionen des ParseControllers

[59] Vgl. jsonld. js, https://github.com/digitalbazaar/jsonld.js (12.09.2016)
[60] Vgl. jsonld-rdfa-parser, https://github.com/scienceai/jsonld-rdfa-parser (12.09.2016)

Während der Implementierung wurde festgestellt, das es nicht möglich ist, die Attribute mit einer Annotation wie “@id” als Attribut in einem Mongoose-Schema zu verwenden. Daher musste dieser Schlüssel vor der Benutzung des Modells umgewandelt und das “@” entfernt werden. Bei der Ausgabe der Daten muss dieser Prozess dann wieder rückgängig gemacht werden.

Der Kontext soll nicht in die Datenbank eingetragen werden und muss deswegen vorher entfernt und dementsprechend vor der Rückgabe an den Client wieder angehängt werden, da die Daten mit ihren semantischen Informationen ausgeliefert werden sollen. Wie in Listing 5 zu sehen ist, übernehmen die Funktionen *convertToJSON* und *removeAnnotations* diese Aufgaben. Für eine Rückwandlung der Daten gibt es die Funktionen *addContext* und *addAnnotations*.

```
//remove context and annotation
function convertToJSON(obj, callback) {
    callback = callback || function() {};

  //remove annotation
  var data = obj['@graph'];
  if(data === null){
       return callback("No Data found");
  }
  removeAnnotations(obj);
  return callback(null, obj);
}

var annotations = [["@id", "id"],["@type","type"],
                   ["@language","language"],["@value","value"]];

function removeAnnotations(obj){
  for (var i = 0; i < obj.length; i++) {
       var temp = obj[i];
       for(var j=0; j<annotations.length; j++){
       temp[annotations[j][1]] = temp[annotations[j][0]];
       delete temp[annotations[j][0]];
       }
  }
}
```

Listing 5: Entfernung des Kontexts und der Annotationen

**locModel** (*lib/models/locModel.js*)
In der Datei wird ein *Mongoose* Schema definiert, das wie im Entwurf beschrieben die verschiedenen Attribute des InLOC Modells abbildet, inklusive der Assoziationen. Das Schema entspricht somit dem JSON-LD Kontext. Die Datentypen sind entweder Strings, Date oder JavaScript Objekte, um die Sprach-Container abzubilden. Es wird hierbei nur der Key “en” verwendet, da der Prototyp eine Mehrsprachigkeit momentan nicht unterstützt. Bei einer Weiterentwicklung sollte ein Weg gefunden werden, wie man die verschiedenen Ländercodes leicht in das Schema einbinden kann. Möglich wäre dies über ein enum. Die Validierung des LOC-Typs erfolgt auch durch ein enum, bei dem die Werte “LOCstructure” und “LOCdefinition” akzeptiert werden. Alle Felder bis auf das *id* Feld sind optional und müssen somit nicht vorhanden sein. Das *id* Attribut muss außerdem einmalig in der Kollektion sein.

## 5.2 Implementierung der Client Applikation

Bei dem User Interface des Prototypen handelt es sich um eine kompakte Anwendung mit nur vier Seiten, die über eine REST Schnittstelle mit dem Server kommuniziert, solch eine Anwendung wird als Single-Page-Webanwendung[61] bezeichnet.

Die JavaScript Bibliothek AngularJS vereinfacht die Erstellung solcher Single-Page-Webanwendung und wurde für die Implementierung der Client-Applikation benutzt. In AngularJS wird ein clientseitiges Model gemeinsam mit der Logik in einem Controller definiert. Die Controller werden anschließend zu einem Modul zusammengefasst. Die Module werden mit Hilfe eines integrierten Dependency-Injection-Containers in die Applikation eingebunden. Dabei wird die View mit dem Model verbunden. Diese Datenbindung ist bidirektional, das heißt, Benutzereingaben wirken sich auf das Model aus, programmatische Änderungen am Model aber auch auf die Benutzeransicht aus.[62]

Die Client-Applikation wird beim Aufruf des Wurzelverzeichnisses an den Client übertragen. Der *public* Ordner des Servers ist öffentltich und enthält alle Dateien, die die Client- Applikation bilden. Im Folgenden sollen nun die einzelnen Klassen und Elemente der Applikation vorgestellt werden.

**index** (*index.html*)
Die Index-Datei erhält der Benutzer bei Abruf der Internetadresse, auf der das Verzeichnis gehostet wird. Innerhalb der Webseite werden sämtliche Skripte wie Bootstrap, jQuery[63], AngularJS vom Server abgerufen und geladen wie auch die selbst erstellten Module.

Mit der Angular Direktiven *ng-app=”compRepoApp”* wird die Applikation eingebunden und mit *ng-view* wird der Container bestimmt, in dem die verschiedenen Ansichten (engl. *views*) eingebunden und ausgetauscht werden. Die gesamte Interaktion mit dem Benutzer findet letztendlich in dieser einen Webseite statt, allerdings werden dabei permanent die Inhalte ausgetauscht. Die Datei enthält außerdem die Navigation der Seite

**app** (*app.js*)
In dieser Datei wird die eigentliche Applikation definiert. Da alle Controller und der Service jeweils in ein eigenes Modul ausgelagert sind, wird nur die Applikation initialisiert und sämtliche Module injiziert. Über einen gemeinsamen Namenraumes können dann alle Module aufgerufen werden.

[61] Vgl. Single-page-Webanwendung, https://de.wikipedia.org/wiki/Single-page-Webanwendung (10.09.2016)
[62] Vgl. AngularJS, https://www.angularjs.org/ (10.09.2016)
[63] Vgl. Javascript Bibliothek jQuery, https://jquery.com/ (10.09.2016)

**appRoutes** (*appRoutes.js*)
Über die definierten Routen kann auf die verschiedenen Seiten navigiert werden. Dabei wird für jede Route ein Template spezifiziert und gegebenenfalls auch ein Controller, der das Verhalten der Ansicht steuert. Es gibt insgesamt vier Routen, die auf verschiedenen Seiten weiterleiten. Die Routen entsprechen dem Entwurf, zusätzlich wurde noch eine Route *test* festgelegt.

**ApiService** (*services/ApiService*)
Das Modul steht als Service allen Controllern zur Verfügung, um Anfragen an den Server zu stellen. Wie man in Listing 6 sieht, wird In den Methoden das Angular-Modul *http* verwendet, welches Http-Methoden bereit stellt um mit dem Server zu kommunizieren. Die Funktionen entsprechen dabei der REST API des Servers.

```
//search
var search = function(query, callback){
    $http({
    method: 'POST',
    url: '/search',
    data: {"query" : query}
    })
    .success(function(data){
    callback(null, data);
    })
    .error(function (error, status){
    callback(error);
    });
};
```

Listing 6: Search Funktion des ApiService

**ScrapeCtrl** (*controllers/ScrapeCtrl*)
Der ScrapeCpntroller verfügt nur über eine Funktion, die eine übergebene URL an den Server weiterleitet und eine Antwort in Form einer Nachricht oder eines Fehlers erhält und diese dann entsprechend ausgibt.

**SearchCtrl** (*controllers/SearchCtrl*)
Das Eingabefeld für die Suche, bindet über die Direktive *ng-model* den Wert des Textfeldes an ein Model, auf das dann im SearchController zugegriffen werden kann. Beim betätigen der Schaltfläche wird die URL im Modell an den ApiService übergeben, der dann eine Http-Anfrage an den Server stellt. Die zurückgelieferten Ergebnisse werden auch wieder im Modell gehalten und füllen eine Liste. Die einzelnen Einträge werden dabei mit der Direktive *ng-repeat* in die Liste eingetragen. Wählt der Benutzer über einen Klick eines der Listenelemente aus, so öffnet sich ein Fenster, in dem die Details einer Kompetenz dargestellt werden. Listing 7 zeigt die Initialisierung des Fensters.

```
$scope.showModal = function(id) {

       ModalService.showModal({
          templateUrl: "../../views/modal.html",
          controller: "ModalCtrl",
          inputs: {
                     id: id
       }
       }).then(function(modal) {
          modal.element.modal();
          modal.close();
       });
};
```

Listing 7: Methodenaufruf zum öffnen eines Fensters

Der Vorteil eines Fensters gegenüber dem Aufruf einer neuen Seite, ist die Erhaltung des Suchergebnisses. Da sich das Fenster einfach über die aktuelle Seite legt. Für die einfachere Benutzung wurde das externe Modul *angular-modal-service*[64] benutzt. Das Fenster besitzt einen eigenen Controller, der das Verhalten des Fensters steuert. Dem *ModalController* wird die *id* der ausgewählten Kompetenz übergeben, bei einem Aufruf des Fensters wird die, in Listing 8 angeführte *loadData* Funktion aufgerufen, die alle IDs der Assoziationsfelder von dem Server abfragt und die Ergebnisse in ein lokales Modell einträgt. Die Wissens-Beispiele und Leistungsniveau-Stufen einer Kompetenz können so in Form einer Liste angezeigt werden.

[64] Angular Modal Service, https://github.com/dwmkerr/angular-modal-service (10.09.2016)

```
$scope.loadData = function() {
        ApiService.competence($scope.id, function(result){
              $scope.model = result[0];

              $scope.id = $scope.model.id;
              $scope.title = $scope.model.title.en;
              $scope.description = $scope.model.description.en;
              $scope.parts = $scope.model.hasLOCpart;
              $scope.examples = $scope.model.hasExample;
              $scope.levels = $scope.model.hasDefinedLevel;


              //hasExample
               ApiService.competence($scope.examples, function(result2){

              $scope.examples = result2;
              });

              //hasLOCpart
               ApiService.competence($scope.parts, function(result){
              $scope.parts = result;
              });

              //hasDefinedLevel
               ApiService.competence($scope.levels, function(result3){

              $scope.levels = result3;
              });
    });
};
```

Listing 8: Die load-Funktion des ApiService

**Layout und Design**

Das Design spielte für den Prototypen eine untergeordnete Rolle, um jedoch eine einfaches, solides Aussehen zu erhalten, wurde die Bibliothek *Bootstrap* benutzt. Durch Angabe von bestimmten Klassenbezeichnungen kann auf vordefinierte Stile und Komponenten zurückgegriffen werden. Durch das von Bootstrap vorgegebene Grid-System, passt sich die Webseite außerdem responsiv an die Display-Größe des Gerätes an.[65]

[65] Vgl. Bootstrap, https://getbootstrap.com/ (10.09.2016)

# 6. Ergebnisse & Auswertung

Im Rahmen dieser Arbeit wurde ein Software-Prototyp entwickelt, der es ermöglicht, eine Website nach Kompetenz-Rahmenwerk-Metadaten zu durchsuchen, diese Daten in eine Datenbank einzutragen und über eine Frontend-Applikation durchsuchbar zu machen. Der Prototyp erfüllt alle in Kapitel 3 definierten Anforderungen. In diesem Kapitel sollen zunächst die beiden grundlegenden Funktionen im Detail beschrieben und in Abschnitt 6.3 die Erweiterungsmöglichkeiten des Prototypen aufgezeigt werden.

## 6.1 Demonstration des Prototypen

Das Erfüllen der Anforderungen durch den Prototypen soll exemplarisch an den in Kapitel 3.3 definierten Anwendungsfällen demonstriert werden. Dabei wird auf die Ausführungen sowohl im Backend als auch im Frontend eingegangen. Im Folgenden werden das Zufügen eines Eintrages in das Verzeichnis sowie die Suche nach bestimmten Kompetenzen innerhalb des Verzeichnisses dargestellt.

### 6.1.1 Hinzufügen eines Eintrags

Der Benutzer gibt eine URL in das Eingabefeld ein. Falls der eingegebene Text nicht dem gültigen Format einer URL entspricht, wird er auf den Fehler aufmerksam gemacht. Diese Funktion stellt HTML5 bereit, wenn der entsprechende Input-Typ auf URL gesetzt ist. Nach Betätigen der Schaltfläche wird die URL über ein POST-Request an den Server gesendet. Auf Serverseite wird die URL an den Parser weiter geleitet. Der Parser ruft die Webseite auf und extrahiert die semantischen Daten, die im RDFa Format bereitgestellt sind. Anschließend gleicht er die Daten mit dem InLOC Kontext ab und validiert damit die Verwendung des richtigen Vokabulars. Die Daten werden ohne Kontext in die Datenbank geschrieben. Außerdem werden die Annotationen von *@id* und *@type* entfernt, da MongoDB diese nicht verarbeiten kann. Das übernimmt der *parseController*. Die Daten werden über Mongoose in die Datenbank geschrieben und vorher anhand des Schemas validiert. Waren alle Schritte erfolgreich, erhält der Benutzer eine Erfolgsbestätigung. Ist bei einem der Schritte ein Fehler aufgetreten, so wird eine Fehlermeldung zurückgeliefert.

### 6.1.2 Suche

Der Benutzer gibt einen Suchbegriff in das Eingabefeld ein und sendet diesen durch Betätigung der Schaltfläche an den Server. Es wird in den Attributen *description*, *title* und *abbr* (Abkürzung) gesucht und die entsprechenden Datenbankeinträge werden zurückgeliefert.

Bei der Datenbankabfrage wurde dahingehend gefiltert, ob der Eintrag ein Titel-Attribut besitzt. Damit soll sichergestellt werden, dass nur Kompetenzen zurückgeliefert werden und nicht etwa Wissens-Beispiele oder Leistungsniveau-Stufen, da diese nur Elemente einer Kompetenz sind und innerhalb dieser dargestellt werden sollen.

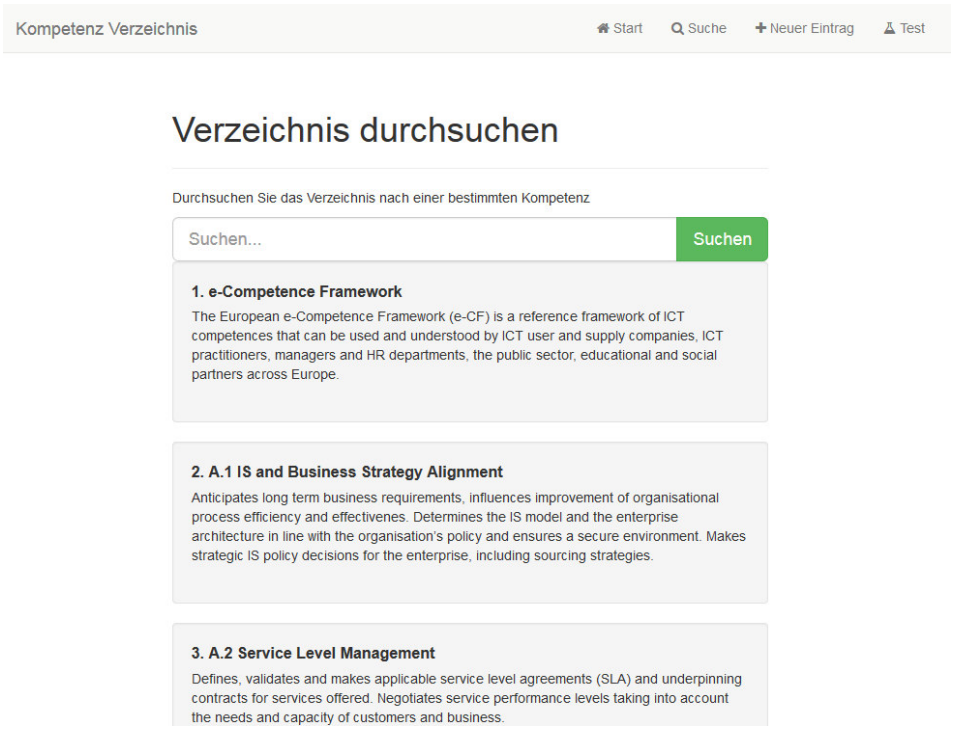


Abbildung 12: Screenshot der Client-Applikation, Suchergebnis

Die Suchergebnisse werden in einer Listenansicht ausgegeben, in der jedes der Elemente einen Titel und eine Beschreibung enthält (vgl. Abbildung 12). Bewegt der Benutzer die Maus über eines der Listenelemente, erfolgt ein Farbwechsel, womit angezeigt wird, dass der Eintrag durch einen Klick ausgewählt werden kann. Wird ein Suchergebnis angeklickt, öffnet sich ein Fenster, das sämtliche Informationen zu der ausgewählten Kompetenz anzeigt (vgl. Abbildung 13). Die Attribute, die eine Assoziation darstellen, werden erst beim Öffnen des Fensters geladen und in einer eigenen Listenansicht angezeigt.

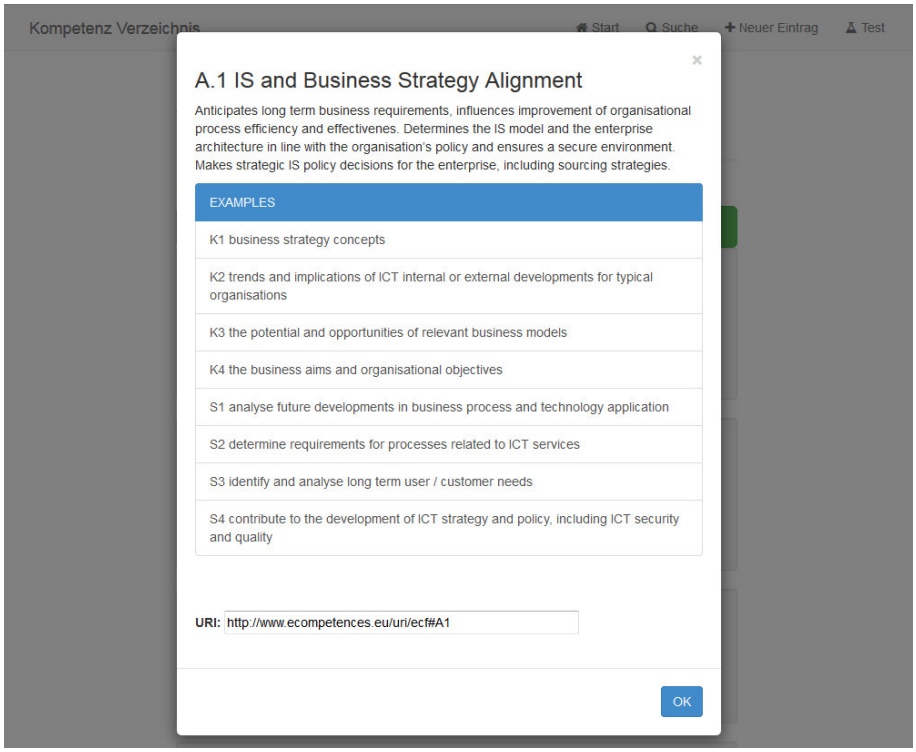

Abbildung 13: Screenshot der Client-Applikation, Detailansicht

Die URI der Kompetenz wird ebenfalls angezeigt und kann vom Benutzer kopiert werden, um an anderer Stelle, wie zum Beispiel innerhalb eines Open Badges, darauf zu verlinken.

## 6.2 Schwierigkeiten

Es gab zwei wesentliche Schwierigkeiten, die eine Implementierung erschwert haben.

**Abstraktionslevel des InLOC Modells**
Die Struktur der Daten ist aufgrund der Verwendung des InLOC Modells sehr allgemein gehalten. Das ist zwar gut für die Abbildung verschiedenster Kompetenzrahmenwerke, allerdings führt das zu einem Problem, wenn man die verschiedenen Elemente und Attribute grafisch aufbereiten und unterschiedlich darstellen will. So sind im Falle des e-CF sowohl Areas, Kompetenzen, Wissensbeispiele als auch Leistungsniveaustufen alles LOCdefinitionen. Es muss ein Weg gefunden werden, um diese verschiedenen Typen zu unterscheiden. Die Wissensbeispiele und Leistungsniveaustufen werden im Prototyp aus dem Suchergebnis herausgefiltert. *Mongoose* erlaubt es der *find* Methode ein Objekt zu übergeben, in dem festgelegt werden kann, dass nur Einträge zurückgeliefert werden, die ein bestimmtes Attribut besitzen. Wie in Listing 9 zu sehen ist, wird dafür die Variable *$exists* benutzt. Um im Suchergebnis nur Kompetenzen anzuzeigen, werden die Einträge mit dem Attribut *hasExample* und *hasDefinedLevel* bei der Datenbankabfrage ausgeschlossen. Das stellt zwar keine optimale Lösung dar, löst jedoch das Problem.

```
LOC.find({
    hasExample: {$exists: false},
    hasDefinedLevel: {$exists: false}
 }
...
```

Listing 9: Die find-Methode filtert verschiedene LOC-Elemente

**URIs als Identifikator**
Ein weiteres Problem stellt das Auflösen der Referenzen, also der Assoziationen, dar. Ein LOC wird anhand seiner ID abgerufen, allerdings handelt es sich hierbei um einen String, der eine URI repräsentiert. MongoDB arbeitet intern aber mit einem Identifikator, der sogenannte ObjectID, der aus einem 12 Byte Hexadezimal Wert besteht.[66] Die ObjectID wird auch verwendet, wenn ein Objekt in der Datenbank ein anderes Objekt anhand seiner ID referenziert. Wenn ein LOC Objekt zum Beispiel mehrere IDs in seinem hasLOCpart Attribut besitzt, könnten diese IDs automatisch durch die eigentlichen LOC Objekte ersetzt und so an den Client zurückgeliefert werden. Für diese Auflösung der IDs stellt MongoDB die Methode *populate* zur Verfügung, jedoch mit dem Problem, dass diese Methode nur mit ObjectIDs funktioniert und nicht mit Strings. Eine Auflösung der IDs muss deshalb manuell erfolgen.

Im Prototyp wurde der Ansatz gewählt, dass die IDs erst bei dem Aufrufen einer Kompetenz aufgelöst werden. Dabei findet ein erneuter Aufruf der Datenbank statt, bei dem alle IDs des Assoziationen-Feldes übergeben werden. Die entsprechenden Objekte werden zurück- geliefert und in einer Liste innerhalb der Kompetenzdarstellung visualisiert.

[66] Vgl. MongoDB ObjectId, https://docs.mongodb.com/manual/reference/method/ObjectId/

## 6.3 Erweiterungsmöglichkeiten

Die in Kapitel 3.3 und 3.4 definierten Anforderungen werden von dem Prototypen erfüllt, stellen jedoch nur die grundlegenden Funktionen dar. Vor einem professionellen Einsatz muss der Prototyp noch um verschiedene Funktionen erweitert werden. Einige dieser Funktionen und andere Ansätze zur Weiterentwicklung sollen in diesem Abschnitt angesprochen werden.

**Authentifizierung**
Wie bei den Anwendungsfällen in Kapitel 3.2 bereits erwähnt, verfügt der Prototyp über keine Authentifizierung. Im derzeitigen Zustand kann jeder Benutzer sowohl in dem Verzeichnis suchen als auch Einträge in die Datenbank veranlassen. Bei einem tatsächlichen Einsatz wäre das nicht erwünscht, sondern es sollte nur registrierten Benutzern mit besonderen Rechten vorbehalten sein. Das Durchsuchen des Verzeichnisses sollte jedoch auch in Zukunft für jeden Benutzer ohne Registrierung möglich sein. Eine Erweiterung müsste also eine Form der Authentifizierung integrieren.

**Feedback vor Persistierung**
Die semantischen Daten werden momentan direkt in die Datenbank eingetragen. Besser wäre aber eine Rückmeldung des Systems nach der Analyse. So könnten zum Beispiel die extrahierten Daten in geeigneter Form präsentiert werden und der Benutzer könnte dann die Möglichkeit erhalten, einem Eintrag in die Datenbank zuzustimmen.

**Mehrsprachigkeit**
Die Text enthaltenden Attribute, wie *title* und *description*, können mehrsprachig sein. Daher ist im Kontext ein Sprachcontainer als Typ definiert. Der verwendete RDFa Parser liest automatisch die definierte Sprache der Webseite aus und trägt sie in das JSON-LD Dokument ein. In der Client-Applikation wird eine Mehrsprachigkeit bisher nicht berücksichtigt, sondern es wird angenommen, dass die Strings in englsicher Sprache vorliegen, also mit dem Schlüssel “en” eingetragen sind. Um in Zukunft verschiedene Sprachen zu unterscheiden, muss der Code der Client-Applikation angepasst werden.

**Darstellung der Daten**
Der Prototyp stellt die abgerufenen Daten auf sehr einfache Weise dar. So kann man zum Beispiel nur einzelne Kompetenzen in einer Detailansicht anzeigen lassen, es ist aber nicht möglich, von einem Element des Rahmenwerkes direkt auf ein anderes zu springen. Eine verbesserte Darstellungsform, die auch die Navigation innerhalb des Netzwerkes von Kompetenzen unterstützt, wäre in Zukunft wünschenswert.

**Datenstruktur**
Frontend verlässt sich darauf, dass Daten eine bestimmte Struktur haben oder über eine bestimmtes Attribut verfügen. So wird z.B. angenommen, dass jedes Wissensbeispiel über die inverse *isExampleOf* verfügt, um diese beim Sucherebnis herauszufiltern. Diese Inverse müsste jedoch nicht zwingend angegeben werden, um das InLOC-Modell einzuhalten. Solche Regeln müssen aber definiert werden, bzw. wenn es ein Programm gibt, dass diese Annotationen setzt, dann müssen solche Regeln implementiert werden.

**Webcrawler und Aktualisierung**
Der Prototyp benutzt einen Parser um die semantischen Daten aus einer Webseite zu extrahieren. Er folgt dabei aber nicht den angegebenen URIs um diese gegebenenfalls auch in die Datenbank einzutragen. In dem Fall, dass ein Rahmenwerk mit allen seinen Elementen innerhalb einer Webseite definiert ist und außerdem keine Verknüpfungen zu einem anderen Rahmenwerk bestehen, stellt das kein Problem dar. Bei einer Aufteilung des Rahmenwerkes auf mehrere Seiten oder einer Verlinkung zu anderen Domains, müssten aber alle Seiten einzeln eingelesen werden. Da das sehr unpraktikabel ist, sollte in einer Erweiterung des Prototypen ein *Webcrawler*[67] zum Einsatz kommen. Diese Art Programm würde den angegebenen URLs automatisch folgen und die gefundenen Daten in die Datenbank eintragen, falls diese noch nicht eingefügt wurden sind. Ein weiterer Vorteil wäre eine automatische Aktualisierung der Daten, in dem das Netzwerk der verlinkten Kompetenzen erneut automatisch durchsucht wird. Eine Aktualisierung kann dabei über den Vergleich der Versionsnummer eines Elementes initiiert werden, wofür das InLOC-Modell das Attribut *version* bietet.

[67] Vgl. Webcrawler, https://de.wikipedia.org/wiki/Webcrawler (20.09.2016)

# 7. Zusammenfassung & Ausblick

## 7.1 Zusammenfassung

Im Rahmen dieser Abschlussarbeit ist ein Prototyp für ein digitales Kompetenz-Verzeichnis entstanden. In Form eines Webservice wurden sowohl eine Server-Anwendung als auch eine Client-Anwendung implementiert. Das Backend besteht aus einem *Node.JS* Server, der auf eine *MongoDB* Datenbank zugreift und dort die Kompetenzdefinitionen in Form von JSON Dokumenten speichert. Über eine REST-Schnittstelle kann der Client Suchaufträge an den Server leiten sowie URLs übermitteln, um neue Einträge in die Datenbank einzufügen. Dafür wird die spezifizierte Webseite von einem Parser nach semantischen Metadaten durchsucht, die dem InLOC Vokabular entsprechen. Sind die Daten valide, werden sie in die Datenbank eingetragen und stehen bei weiteren Suchanfragen zur Verfügung. Das Frontend wurde in Form einer *Single-Page-Webanwendung* mit der JavaScript Bibliothek *AngularJS* umgesetzt.

Die zugrunde liegenden Konzepte, auf denen das implementierte Kompetenz-Verzeichnis basiert, wurden bereits in Kapitel 3.1 zusammengefasst . Der Prototyp sollte dabei vor allem zeigen, dass das vorgestellte Konzept aus der Verwendung des InLOC Standards in Kombination mit einem Verzeichnis, welches die semantischen Daten automatisch aus einer Webseite extrahiert und in eine Datenbank einträgt, gut funktioniert.

Wie in Kapitel 6.3 dargestellt wurde, gibt es noch einige Erweiterungsmöglichkeiten, um den Prototypen zu verbessern bevor er in einem professionellen Umfeld eingesetzt werden kann. Der Prototyp stellt nur eine Variante einer Implementierung dar und wie in Kapitel 6.2 aufgezeigt wurde, sind dabei auch einige Schwierigkeiten aufgetreten.

Einerseits führte die verallgemeinernde Abstraktion des InLOC-Modells zu Schwierigkeiten mit der Darstellung der verschiedenen Elemente einer Kompetenzdefinition, andererseits stellte die Verwendung von URIs als Identifikator in der Datenbank ein Problem dar. Diese Probleme konnten zwar umgangen werden, die Implementierung stellt in diesen beiden Punkten aber nicht unbedingt die optimale Lösung dar. Bei einer Weiterentwicklung des Prototypen könnte zum einen über die Verwendung einer relationalen Datenbank nachgedacht werden, zum anderen könnte versucht werden, eine Möglichkeit zu finden, wie die einzelnen Elemente des InLOC-Models besser voneinander unterschieden können.

## 7.2 Ausblick

Der Prototyp eines Kompetenz-Verzeichnisses hat gezeigt, dass das InLOC Modell für die Abbildung von Kompetenz-Rahmenwerken geeignet ist und dass das Einlesen von semantisch annotierten Kompetenzdefinitionen aus einer Webseite gut funktioniert. Für eine tatsächliche, praktische Anwendung müssen die semantischen, verlinkten Daten in die Webseiten von Kompetenz-Rahmenwerken eingebunden werden. Erst wenn der Großteil der Anbieter das tut, macht die Verwendung des Kompetenz-Verzeichnisses Sinn.

Für diesen Zweck müsste eine zweite Anwendung entwickelt werden, die es erlaubt, die semantischen Metadaten automatisch zu generieren. Über eine geeignete grafische Schnittstelle könnte so ein Benutzer, die erforderlichen Informationen, die eine Kompetenz oder ein Kompetenz Rahmenwerk beschreiben, in ein Formular eintragen. Das Programm würde dann ein RDF-konformes Dokument, mit allen semantischen Metadaten erstellen. Eine solche Anwendung wäre für die Beschreibung bzw. Anreicherung eines Kompetenz-Rahmenwerkes mit semantischer, verlinkter Information in Form des InLOC-Modells, unabdingbar, da das manuelle Erstellen einer solchen Datei zu umständlich ist und von der Benutzung dieser Methode abschreckt.

Eine breitere Akzeptanz könnte das Konzept auch erfahren, wenn das InLOC-Vokabular in eine Erweiterung des schema.org Vokabulars überführt werden würde. Wie in Kapitel 2.1.10 angesprochen, ist das schema.org Vokabular wesentlich weiter verbreitet als die individuelle Lösung des InLOC Vokabulars.

Die Weiterentwicklung des Prototypen hin zu einer professionellen Anwendung sowie die Überführung der Kompetenz-Rahmenwerke in eine semantische, verlinkte Form, würde die Verwendung von Open Badges im Bildungs- und Wirtschaftsbereich mit Sicherheit erhöhen. Das InLOC Informationsmodell bietet sich dabei als möglicher universeller Standard an, um Kompetenz-Rahmenwerke abzubilden.

# A. Anhang

## CD-Inhalt

| Datei | Inhalt |
|---|---|
| Competence-Repository (Ordner) | Sourcecode der Umsetzung, Competence-Repository |
| Readme.txt | Installations-/Verwendungsanleitung |
| RDFa (Ordner) | Ausschnitt des e-CF in RDFa |
| Bachelorarbeit.pdf | Digitale Version der Bachelorarbeit |

# B. Abkürzungsverzeichnis

| | |
|---|---|
| **API** | Application Programming Interface |
| **Cetis** | Centre for Educational Technology, Interoperability and Standards |
| **e-CF** | e-Competence Framework |
| **EQF** | European Qualifications Framework |
| **ICT** | Information and Communication Technology |
| **InLOC** | Integrating Learning Outcomes and Competences |
| **IRI** | Internationalized Resource Identifier |
| **JSON** | JavaScript Object Notation |
| **LOC** | learning outcome or competence |
| **JSON-LD** | JSON-basierte Serialisierung für verLinkte Daten |
| **RDF** | Resource Description Framework |
| **RDFa** | Resource Description Framework in Attributes |
| **REST** | Representational State Transfer |
| **URI** | uniform resource identifier |
| **W3C** | World Wide Web Consortium |

# C. Literaturverzeichnis

[Bad15] Badge Alliance: *Open Badges Technical Specification*. Stand: 01.05.2015.
URL: https://openbadgespec.org/ (letzter Abruf am 10.09.2016)

[Buc16] Buchem, Ilona u.a.: *Open Badge Network Discussion Paper on Open Badges at Policy Levels. Outcome 05-A1 - Discussion Paper on Open Badges at Policy Levels* - Stand: Juli 2016.
Abrufbar unter:
http://www.openbadgenetwork.com/wp-content/uploads/2016/01/OBN-O5-A1-Policy-Discussion-Paper-31-July-2016.pdf (letzter Abruf am 18.08.2016)

[Cena13] CEN: *InLOC - Part 1: Information Model for Learning Outcomes and Competences*. Stand: Juli 2013. Abrufbar unter: ftp://ftp.cen.eu/CEN/Sectors/List/ICT/CWAs/CWA%2016655-1.pdf (letzter Abruf am 15.08.2016)

[Cenb13] CEN: *InLOC - Part 2: Guidelines including the integration of Learning Outcomes and Competences into existing specifications*. Stand: Juli 2013.
Abrufbar unter: ftp://ftp.cen.eu/CEN/Sectors/List/ICT/CWAs/CWA%2016655-2.pdf (letzter Abruf am 15.08.2016)

[Cen14] CEN: *European e-Competence Framework 3.0. A common European Framework for ICT Professionals in all industry sectors*. Stand: 2014. Abrufbar unter:
http://relaunch.ecompetences.eu/wp-content/uploads/2014/02/European-e-Competence-Framework-3.0_CEN_CWA_16234-1_2014.pdf (letzter Abruf am 20.08.2016)

[Gra16] Grant, Simon - *InLOC: the potential of competence structures*, Mai 2013,
http://de.slideshare.net/asimong/inloc-the-potential-of-competence-structures (letzter Abruf 10.09.2016)

[Hit08] Hitzler, Pascal/ Krötsch, Markus/ Rudolph, Sebastian.: *Semantic Web - Grundlagen, Aufl. 1,* Berlin Heidelberg: Springer Verlag, 2008.

[Kel16] Kellog, Gregg - *JSON-LD and MongoDB.* Stand: 18. Aug. 2012.
URL: http://de.slideshare.net/gkellogg1/jsonld-and-mongodb (letzter Abruf am 10.09.2016)

[Kon16] Konert, Johannes: *Open Badge Network Proposal on Competency Alignment and Directory. Outcome 03-A2 - Competency Directory*. Stand: 25.02.2016

[Mei16] Meier, Andreas/ Kaufmann Michael: *SQL- & NoSQL-Datenbanken, Aufl.* 8, Berlin Heidelberg: Springer Verlag, 2016

[Spo12] Sporny, Manu u.a.: *JSON-LD Syntax* 1.0. Stand: 22.05.2012.
URL: http://json-ld.org/spec/ED/json-ld-syntax/20120522/ (letzter Abruf am 20.09.2016)

[Woo14] Wood, David/ Zaidman, Marsha/ Ruth, Luke: *Linked Data - Structured Data on the Web,* Shelter Island, New York: Manning Publications Co., 2014.

# D. Abbildungsverzeichnis

# E. Tabellenverzeichnis



# F. Listings

**Eidesstattliche Erklärung**

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien – oder Prüfungsleistung war.

Berlin, den 04.10.2016

_____

Christian Mehns