Deep-Crawling-Assistenzsysteme als E-Learning-Autorenwerkzeuge

Masterarbeit

KIT – Karlsruher Institut für Technologie
Fraunhofer IOSB – Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung

Simon Schwarz

14. Mai 2021

Verantwortliche Betreuer:	Prof. Dr.-Ing. J. Beyerer
	Prof. Dr.-Ing. T. Längle
Betreuender Mitarbeiter:	Dipl.-Inf. A. Streicher

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der gültigen Fassung beachtet habe.

Karlsruhe, den 14. Mai 2021

(Simon Schwarz)

Kurzfassung

Suchmaschinen spielen eine essenzielle Rolle in der Nutzung des Internets. Ihre Nutzbarkeit im Kontext des E-Learning ist aber sehr eingeschränkt, denn mit konventionellen Methoden und Technologien sind sie nicht in der Lage Inhalte in Learning Management Systemen zu indizieren und zu durchsuchen. Im Rahmen dieser Arbeit wird eine General-Purpose Deep-Web Suchmaschine für E-Learning-Inhalte entworfen und implementiert. Grundlage der Suchmaschine ist ein LTI-basierter Deep Web-Crawler. Desweiteren werden vor dem Hintergrund der Interoperabilität von Informationssystemen Erweiterungen wie eine semantische Suchfunktionalität und ein Recommender-System diskutiert. Zu diesem Zweck werden Techniken des Natural-Language Processing und des Semantic Web angewendet. Das so entworfene System kommt im Rahmen der am Fraunhofer IOSB entwickelten Common Learning Middleware als Teil eines Kurskurationssystems zum Einsatz. Evaluationsergebnisse zeigen, dass das System in diesem Kontext sehr gut für das Finden von E-Learning-Inhalten geeignet ist, jedoch weniger gut für domänenspezifische Suchen. Entsprechend wird das System als eine modulare Architektur entworfen, die an zukünftige Anwendungsfälle angepasst werden kann.

Abstract

Search Engines play an essential role in how people use the Internet. However, their usefulness in the context of e-learning is severely limited because they are not able to index and search content that is located inside learning management systems using conventional methods. This thesis details the design and implementation of a general-purpose deep web search engine for e-learning content based on a LTI-deep-web crawler. Against the background of the interoperability of information systems additional extensions like semantic search functionality and recommender systems are discussed. To this end, natural language processing and semantic web technologies are leveraged. The designed system is used as part of a course curation system in the Common Learning Middleware developed at the Fraunhofer IOSB. Experimental results demonstrate that the system is well suited to find e-learning content in this context but also somewhat limited for domain-specific information retrieval. As a consequence, the system is designed around a modular architecture which makes fine-tuning to future use cases possible.

Inhaltsverzeichnis

1 Einleitung

Die weiterverbreitete Akzeptanz des Internets hat nahezu alle Bereiche der Gesellschaft und Industrie verändert. Auch das Feld der Bildung zählt dazu. E-Learning, d.h. das Online-Angebot von Lernressourcen wie Texten, Bildern, Videos oder interaktiven Medien ist zum wichtigen Teil von Lehrplänen geworden. E-Learning-Inhalte sind in der Regel in speziell dafür entworfenen Informationssystemen, den sogenannten Learning Management Systemen, organisiert, die extern als normale Webseiten auftreten und als solche per URL aufgerufen werden können. Um die Inhalte zu sehen, muss der Nutzer normalerweise einen Account bei der jeweiligen Institution haben sowie eventuelle Fristen oder rechtliche Einschränkungen beachten. Große Teile der Learning Management Systeme sind also als Teile des Deep-Web anzusehen, das heißt als Webseiten, die nur nach vorheriger Authentifizierung bzw. Autorisierung aufrufbar sind. Eine wichtige Konsequenz davon ist, dass diese nicht von Suchmaschinen indiziert werden. Dies führt zu einem Auffindbarkeitsproblem, denn Suchmaschinen dominieren die Nutzung des Internets. In den meisten Fällen werden nicht explizit URLs oder Webseiten aufgerufen, sondern per Stichworten nach relevanten Inhalten gesucht. Des Weiteren werden fast ausschließlich URLs, die zu den ersten Elementen der Ergebnisliste gehören auch wirklich aufgerufen. Viele E-Learning-Inhalte sind also faktisch nicht nutzbar, selbst wenn ein potentieller Nutzer die Rechte dazu hätte. Das Auffindbarkeitsproblem ist dabei nur eines von zahlreichen Interoperabilitätsproblemen von Learning Management Systemen. Dazu zählen zum Beispiel auch das Fehlen von Standards für das Speichern und Referenzieren von Inhalten oder für den Austausch zwischen Learning Management Systemen unabhängig der jeweiligen Hersteller oder Institutionen.

Ein Ansatz Interoperabilität zu etablieren ist die von IMS entwickelte Technologie *Learning Tools Interoperability* (LTI) [IMS19a]. Bei LTI handelt es sich um einen Standard, der ein Framework definiert, um E-Learning-Inhalte eines LMS in andere zu integrieren. Dabei kommen Deep Links zu Einsatz: URIs, die konkrete Inhalte eines LMS referenzieren. Eine Verwendung dieser Konzepte findet im Rahmen des am Fraunhofer IOSB entwickelten *Common Learning Middleware* (CLM) [IOS21] statt, die das Verwalten von Kursen mit Inhalten aus allen (unterstützten) angeschlossenen LMS in einem zentralen Portal namens Ephesos ermöglicht. Dort werden LMS-übergreifende Kurse mittels XML-Dateien, die die Deep Links enthalten definiert.

1.1 Die Problemstellung

Die CLM stellt zwar einen gewissen Grad an Interoperabilität zwischen E-Learning-Systemen her, bisher ist jedoch das Suchen von Inhalten nicht möglich, die Existenz und Adresse referenzierter Inhalte muss potentiellen Kurs-Autoren und Lernenden bekannt sein. Diese Arbeit untersucht wie die bestehenden Technologien der LTI-Deep Links und der CLM genutzt werden können um das Auffindbarkeitsproblem zu lösen und eine Suchmaschine zu entwickeln, die das Entdecken von E-Learning-Inhalten anhand von Suchbegriffen ermöglicht. Mit der fehlenden Suchfunktionalität einhergehend ist der Fakt, dass das Erstellen der Kurs-Spezifikationen komplett manuell mithilfe von standardmäßigen Text-Editoren erfolgt. Für diese Probleme ist ein Lösung zu entwerfen: Ein Autorenwerkzeug, das das Suchen von Inhalten und deren weitergehende Verwendung ermöglicht. Sekundäre Fragestellungen beziehen sich auf Aspekte der Authentifizierung und entsprechender Sichtbarkeit von Inhalten, den Umgang mit verschiedenen Typen von Inhalten, potentielle Erweiterungen und Ausprägungen des Systems wie die Unabhängigkeit von der CLM, fortgeschrittene Information Retrieval-Funktionalitäten oder Inhaltsvorschläge. Es sind bestehende Technologien wie Web-Crawler oder Such-Bibiliotheken zu verwenden und auf den konkreten Anwendungsfall anzupassen. Neben konzeptionellen Fragestellungen umfasst die Arbeit auch technische und organisatorische Aspekte, insb. die Abstimmung mit Projektpartnern, die Kompatibilität mit einem sich selbst noch in Entwicklung befindlichen System sowie Fragen der Operationalisierung und der Verwendung von Standards.

1.2 Lösungsansatz und Zielsetzung

Das Ziel der Arbeit ist der Entwurf und die Implementierung eines Course Authoring Assistant namens Findoo als Teil des bestehenden CLM-Systems. Das System ist in Abbildung 1.1 dargestellt. Grundlage dessen ist ein Index, der in den via CLM verbundenen LMS verfügbaren Inhalte speichert und verwaltet. Diese sind via LTI-Deep Links identifiziert. Um diesen Index aufzubauen und aktuell zu halten wird ein Deep Web-Crawler benötigt, der Interoperabilitätsstandards nutzt, um die LMS zu durchsuchen. Hier sind zahlreiche Detail-Fragen bezüglich der Rolle der LTI-Komponenten und wie mit der fast beliebigen internen Struktur der LMS umgegangen wird zu beantworten. Ein wichtiger Teil der Interoperabilität ist zudem die Eigenschaft der Robustheit. Das Crawling muss auch bei kleineren Änderungen der LMS funktionieren. Die Nutzer-Interaktion mit dem System soll über eine Web-Applikation stattfinden, deren zwei Hauptkomponenten sind die Suchmaschine auf dem Index und die Editor-Funktionalität. Für letztere soll der Common Cartridge-Standard [IMS15] verwendet werden. Dieser muss im Rahmen der Arbeit also auf die indizierten Inhalte abgebildet und für den Nutzer verständ-

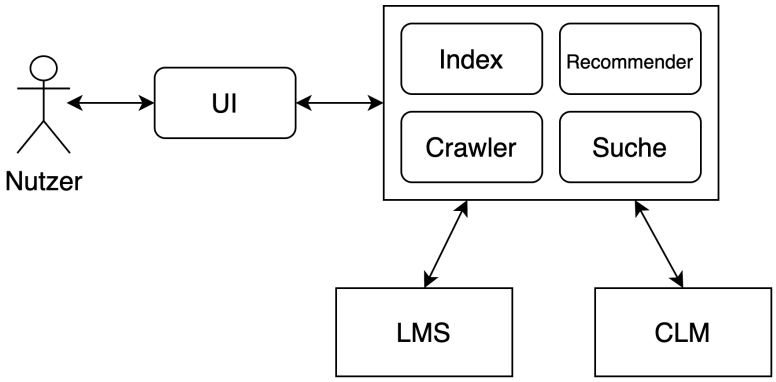


Abbildung 1.1: Eine grobe Übersicht über das zu entwerfende System. Dieses muss mit dem Nutzer, Learning Management Systemen und der CLM interagieren.

lich aufbereitet und dargestellt werden. Die mit der Suche verbundenen Herausforderungen beziehen sich auf methodologische Aspekte wie den Entwurf eines Suchalgorithmus mit entsprechender Relevanzberechnung und die geeignete Repräsentation der Inhalte. Ebenso soll das Filtern von Ergebnissen nach Kriterien wie Typ oder Erfassungszeit möglich sein. Außerdem muss eine Feature-Konstruktion, -Auswahl und -Speicherung stattfinden, um diese Anforderungen an die Suchmaschine zu erfüllen. Ein weiterer Aspekt der Arbeit ist die Erweiterung des Information Retrieval Systems um eine semantische Suchkomponente, die Dokumente aufgrund ihres Inhaltes findet, auch wenn die eingegebenen Stichworte nicht vorkommen. Diese wird durch Techniken des Natural Language Processing realisiert, konkret werden dafür Semantische Einbettungen und Ontologien verwendet. Zusätzlich wird ein Recommender-System für E-Learning-Inhalte integriert, das von den selben Techniken gebraucht macht. Eine zentrale Herausforderung ist hierbei die *General-Purpose*-Eigenschaft des entworfenen Systems. Die Anwendungsdomäne ist nicht a-priori bekannt, folglich können keine klassischen Vorberechnungen wie das spezifische Trainieren von Modellen stattfinden. Da Findoo zu keinem System in der Literatur direkt vergleichbar ist, befolgt die abschließende Evaluation des Systems ein eigens entworfenes Konzept, das die einzelnen Komponenten des Systems in Isolation prüft.

All diese Probleme müssen im Kontext der Kompatibilität und Interoperabilität mit bestehenden und sich selbst entwickelnden Systemen gelöst werden. Diese Co-Evolution wird deshalb noch als eigene Forschungsfrage aufgegriffen, die zu weitergehenden Anforderungen an das System und den Entwicklungsprozess selbst führt. Das System ist in Kooperation mit Teams aus dem Fraunhofer IOSB und einem Industriepartner zu entwickeln.

1.3 Projektumgebung

Die Abschlussarbeit ist in das Fraunhofer-Kooperationsprojekt Common Learning Middleware (CLM)[IOS21] eingebettet. Das Thema des Projekts ist die Entwicklung von integrierten und verteilten Lernumgebungen durch die Anwendung von Interoperabilität-Standards. Wie aus den Market Analysis Reports von eliterate.us [eli19] hervorgeht ist der Markt von Learning Management Systemen ein sehr dynamischer und vielseitiger, entsprechend wichtig ist die Verwendung von Standards. Es gibt mehrere Organisationen, Konsortien und Initiativen, die sich mit der Entwicklung entsprechender Spezifikationen für solche Standards beschäftigen. Beispiele sind das IMS Global Consortium [Con21], dass Standards wie Learning Tools Interoperability (LTI) [IMS19a] oder Question Tools Interoperability (QTI)[IMS20] entwickelt, sowie die Advanced Distributed Learning Initiative (ADL), die unter anderem den Standard Experience API (xAPI) [Ini17] verwaltet.

Das Projekt untersucht diese aktuellen Standards und setzt diese in einer leichtgewichtigen, einfach zu integrierenden Middleware um, die es ermöglicht die Learning Management Systeme in einer Organisation oder zwischen Organisationen zu verbinden, selbst wenn diese Systemlandschaft marktbedingt sehr heterogen ist. Via eines zentralen Webportals kann auf alle angeschlossenen Systeme zugegriffen werden. Die Architektur dieser Middleware ist webbasiert, offen und folgt dem Paradigma der Serviceorientierung. Eine Folge dieses Designs ist, dass potenzielle Anwender nur die von ihnen benötigt Funktionalität lizenzieren müssen.

Die Möglichkeiten einer solchen Middleware können mithilfe von Anwendungsfällen demonstriert werden, wie etwa die Kopplung eines ILIAS-LMS [Bed+13] mit einem Serious Game, dass Technologien nutzt, die in ILIAS nicht nativ unterstützt werden. Inkompatibilitätsprobleme dieser Art treten durch die angesprochene Heterogenität der Systeme häufig auf, sei es fehlende Nutzerakzeptanz bei der Migration von Inhalten eines Systems in ein anderes oder der erhebliche ad-hoc Aufwand, wenn eine solche technisch zunächst nicht möglich ist, z.B. bei plattformspezifischen Anwendungen. In diesem Zusammenhang ist auch die Integration von modernen Aspekten des E-Learning naheliegend, z.B. Learning Analytics oder KI-Ansätze zu Adaptivität, Personalisierung und Recommender-Systeme. Forschung dieser Art wird auch von der EU gefördert, z.B. im Rahmen von INTUITEL.[1] Ein funktionsfähiger Prototyp der Middleware wurde erstmals 2020 im Rahmen der LEARNTEC[2] vorgestellt.

1 http://www.intuitel.eu
2 https://www.learntec.de/

1.4 Gliederung

Die Arbeit besteht aus sieben Kapiteln. Kapitel 1 ist eine thematische Einleitung und beschreibt die Aufgabenstellung und Projektumgebung. Kapitel 2 gibt einen Überblick über relevante, verwandte Systeme aus der Literatur zu E-Learning und Interoperabilität. Kapitel 3 erläutert zentrale theoretische Grundlagen, die im Rest der Arbeit verwendet werden. Kapitel 4 beschreibt dann den konzeptionellen Entwurf des Assistenzsystems auf einer hohen Abstraktionsebene, anschliessend werden in Kapitel 5 die wichtigsten technischen und algorithmischen Details gegeben. Kapitel 6 evaluiert die Performance der einzelnen Komponenten des Systems via einer Reihe von Kriterien und Metriken. Kapitel 7 fasst schliesslich die Arbeit zusammen, bewertet deren Ergebnisse und gibt einen Ausblick auf weiterführende Arbeiten. Abbildung 1.2 zeigt den konzeptionellen Aufbau der Arbeit.

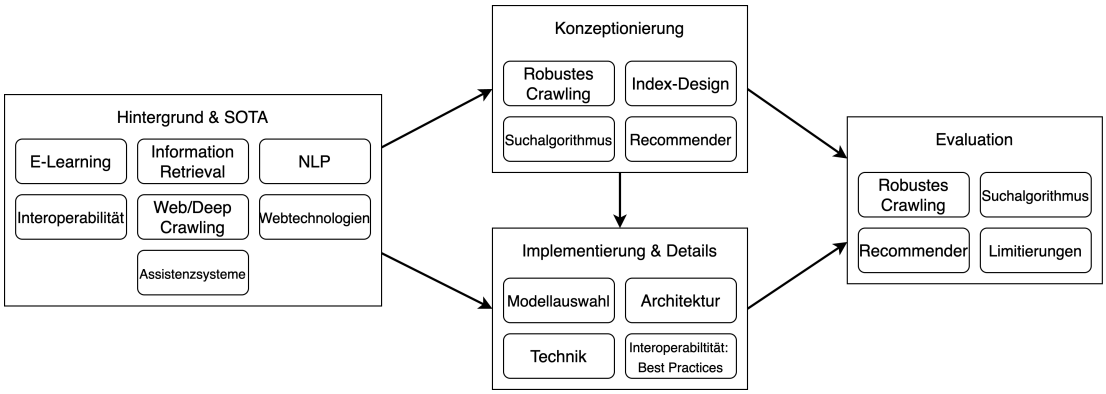


Abbildung 1.2: Der konzeptionelle Aufbau der Arbeit. Diese beginnt mit Kapiteln zum Hintergrund und relevanter Forschung aus der Literatur, diese werden dann für den Entwurf und die Implementierung des Systems benutzt. Die Ergebnisse werden abschließend evaluiert.

2 Stand der Technik

Dieses Kapitel beschreibt aktuelle Entwicklungen und Systeme, die für den Kontext dieser Arbeit relevant sind. Sektion 2.1 beschreibt Werkzeuge zur Erstellung von E-Learning-Inhalten, Sektion 3.3 fasst relevante Arbeiten zur Interoperabilität im E-Learning-Kontext zusammen, Sektion 2.3 beschäftigt sich mit Web-Crawling von E-Learning-Inhalten und Sektion 2.4 schließlich mit Recommender-Systemen für Lerninhalte. Es werden dabei stets die Interoperabilitätsmöglichkeiten der Werkzeuge betont.

2.1 E-Learning-Autorenwerkzeuge

Diese Sektion trägt Autorenwerkzeuge aus der Literatur zusammen. Hierbei muss unterschieden werden zwischen dem *Authoring*, dem Erstellen von E-Learning-Inhalten aus Medien wie Text oder Bildern und der *Curation*, dem gezielten Auswählen und Kombinieren existierender E-Learning-Inhalte. Nur letzteres kommt bei dieser Arbeit zum Einsatz, es sind jedoch die Interoperabilitätsaspekte beider Kategorien relevant.

Kommerzielle Angebote Es gibt zahlreiche kommerzielle Werkzeuge zum Erstellen von E-Lerning-Inhalten, diese sind in der Regel prorietär - Inhalte können zunächst nur in Kursen dieser Tools verwendet werden, oft ist aber auch ein Export gemäß Standards wie SCORM [ADL09] und xAPI [Ini17] möglich. Die bekanntesten zwei Beispiele sind Adobe Captivate [Ado21] und Articulate 360 [Glo21]. Sie bieten beide nahezu identische Funktionalität: Die Werkzeuge erlauben die Erstellung einer Vielzahl von Inhaltstypen, sind dabei aber auf Präsentationen fokussiert, es ist unter anderem möglich PowerPoint-Präsentationen zu importieren und auf deren Basis Navigation, Aufgaben und Videos zu erstellen. Zudem gibt es Editoren für VR-Inhalte. Die erstellten Ergebnisse sind unabhängig von der Hardware-Plattform, aber in vollem Umfang nur in der jeweiligen Software nutz- bzw. editierbar. Zusätzlich existieren Bibliotheken aus Millionen von vorgefertigten Inhalten, die in die Kurse eingebunden werden können [Ado21; Glo21].

Freeware und Open-Source Angebote Neben kommerziellen Angeboten gibt es auch eine Vielzahl von kostenfreien Werkzeugen. Battistella und von Wangenheim [Bato8] evaluieren sechs solcher Tools: *eXe Learning* ist ein browserbasiertes Tool, dass die Erstellung von HTML-Inhalten erlaubt, dazu zählen Text, Multiple-Choice-Aufgaben und Java Applets. *Xerte* ist ebenfalls browserbasiert und wurde speziell für Nutzer ohne Programmierkenntnisse entworfen, es wird großer Wert auf das Einbinden von Ressourcen wie Google Maps oder YouTube-Videos gelegt und der Erstellungsprozess ist auf Wunsch von einem Guide geführt [Bato8]. *CourseLab* ist ähnlich zu den kommerziellen Tools aufgebaut: Es handelt sich um eine native App, die hauptsächlich mit Präsentationen arbeitet und diese auch importieren kann. *HotPotatoes* ist ebenfalls eine Desktop-Applikation, ist aber fokussiert auf Aktivitäten wie Quizzes und Puzzles [Bato8]. *Microsoft LCDS* kombiniert die Eigenschaften der beiden vorangegangenen Werkzeuge und erlaubt die geführte Erstellung von Aktivitäten. *MyUdutu* ist ein cloudbasiertes Produkt, dass die Erstellung von statischen und dynamischen Inhalten erlaubt und als einziges kostenloses Produkt eine Vorlagen-Bibliothek bietet. Alle diese Werkzeuge erlauben den Export erstellter Inhalte via des SCORM-Standards [ADL09; Bato8].

Ediphy Lopez-Pernas et al. [Lop+20] stellen *Ediphy* vor. Sie bemerken, dass es im Allgemeinen schwer ist ein Autoring Tool für alle Zielgruppen und Kontexte zu entwickeln. Eine Lösung für dieses Problem ist durch Modularität gegeben: Entwickler erstellen Plugins, die von normalen Nutzern verwendet werden können. Ein naheliegendes Beispiel sind Plugins für verschiedene Content-Typen wie Audio-, Video-, oder Quiz-Inhalte. Das Editieren folgt dem WYSIWYG-Paradigma [Lop+20]. So können Seiten und Präsentationen erstellt werden, für Layout- und Stylefragen existieren Templates. Die Anordnung von Seiten erfolgt via Drag and Drop in einem automatisch erstellten Inhaltsverzeichnis. Externe Inhalte können auf mehrere Arten wiederverwendet werden, eine Möglichkeit ist das Nutzen von Import-Dialogen und (proprietären) APIs von Vish und diversen Webseiten wie YouTube. Für Moodle-Inhalte wurde ein spezieller Adapter entworfen [Lop+20]. Ein weiteres Feature ist das *Smart Clipboard*, dass durch das Mapping von Mime-Typen ein generalisiertes Copy and Paste ermöglicht. Erstellte Inhalte können in einem proprietären Format exportiert werden und in einer separaten Applikation (dem Ediphy-Viewer) angesehen werden oder als SCORM [ADL09] sowie statisch als PDF [Lop+20].

ViSH Gordillo et al. [GBQ17] stellen einen E-Learning-Editor für den *Virtual Science Hub* (ViSH) vor. Dieser ist als Client-Server-Architektur implementiert und besteht aus dem Vish Editor als Web-App und dem Vish Viewer, der die Ansicht der erstellten Inhalte im Browser ermöglicht. Der Editor ist in Abbildung 2.1 dargestellt. Die Kommunikation zwischen dem

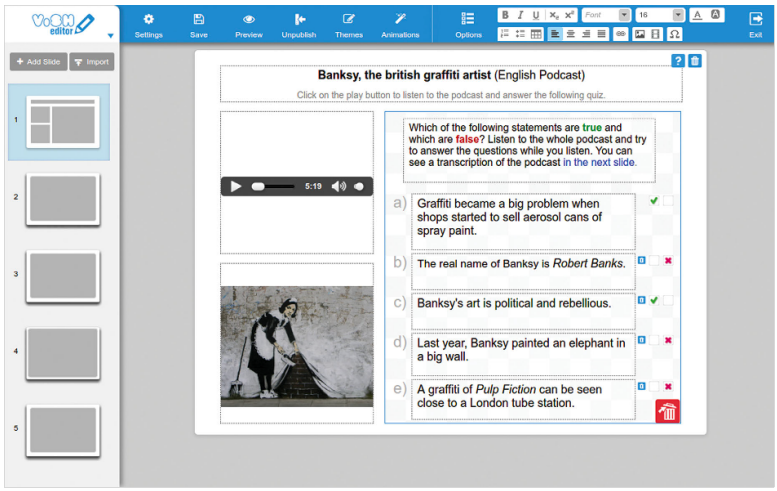


Abbildung 2.1: Das Interface des Vish-Editors [GBQ17].

Editor und dem Server funktioniert über eine API, theoretisch sind also verschieden Implementierungen am Backend für das gleiche Frontend möglich [GBQ17]. Erstellte Objekte werden im JSON-Format repräsentiert und folgen einem eigens definierten Learning Object Model: Objekte sind hierarchisch als Aggregation modelliert. Das Haupt-Objekt besteht dabei aus interaktive Slides, diese sind wiederum modelliert mit den folgenden Ebenen [GBQ17]:

1. Atomare Ressourcen: Bilder, Text bis hin zu Spielen und Websites
2. Slides: Mengen von atomaren Ressourcen
3. Slidesets bestehend aus Flashcards, virtuellen Tours und Enriched Videos
4. Interaktive Präsentationen

Die Modellierung verschiedener Granularitätsebenen muss auch im hier entworfenen System erfolgen.

Elevate Dellagiacoma et al. [Del+20] stellen die Elevate-Suite vor, ein Softwaresystem für die Erstellung von interaktiven Videos, die im Rahmen von *blended* Onlinekursen (also Kurse, die Präsenzelemente mit E-Learning mischen) zum Einsatz kommen sollen. Das Kernelement der Tool Suite ist ein Story-Editor, dargestellt in Abbildung 2.2. In diesem webbasierten Editor kann die Struktur des Videos als gerichteter Graph modelliert und bearbeitet werden - durch Knoten mit assoziierten Videoclips und Übergangsmöglichkeiten zwischen diesen. Die so erstellten Videos können auf einem Server gehostet werden und durch einen Download des entsprechenden Videoplayers und dessen Einbindung als ein SCORM-Paket [ADL09] können diese dann in beliebige andere LMS eingebunden werden [Del+20].

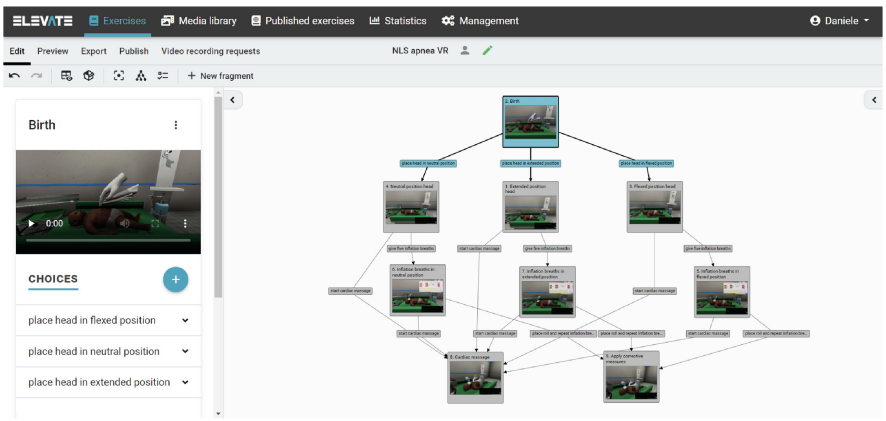


Abbildung 2.2: Das Interface des Elevate-Editors [Del+20].

ONYA Cinici und Altun [ÇA18] entwerfen ein Softwarewerkzeug, das es ermöglicht ein existierendes Repository von E-Learning-Inhalten zu durchsuchen und die gefundenen Inhalte mit IMS-Standards wie Common Cartridge neu zu kombinieren. Als größte Herausforderungen dieses Prozesses wurden rechtliche Belange wie unklare Lizenzen und fehlende Metadaten identifiziert [ÇA18]. Als wichtige Eigenschaften von entsprechenden Autorenwerkzeugen wurde ein einfaches WYSIWYG-Interface und eine Integration von bestehenden Suchmaschinen identifiziert [ÇA18]. Diese Interoperabilitätsaspekte sind auch für das hier zu entwerfende System relevant.

Enook Ahn et al. [Ahn+17] stellen das Autorenwerkzeug Evolutionary Notebook (Enook) vor. Dieses System ist auf die Erstellung von aktivitätsbasierten, d.h. dem learning-by-doing-Paradigma folgende Lernobjekten ausgelegt. Diese sind in ein plattformunabhängiges Ebook-Format integriert. Bei den so erstellten Kursen werden klassische Texte und Bilder mit interaktiven Elementen kombiniert. Diese können bzw. sollen als Teil von Kursen in Schulen und Universitäten verwendet werden und bieten eine Reihe von analytischen und organisatorischen Funktionen. Die Erstellung der Inhalte kann auf verschiedene Weise geschehen, die vom grafischen Interface bis zur Einbindung von JavaScript reichen [Ahn+17]. Eine Folge daraus ist, dass Inhalte nicht exportiert werden können, aber bestehende Inhalte in Formaten wie PDF oder Powerpoint können importiert werden.

LMS Oxford Aldaij und Berri [AB17] entwerfen ein System, das es erlaubt E-Learning-Inhalten aus dem Internet in Lernressourcen zu integrieren, um die Kosten und den Aufwand für deren Erstellung zu reduzieren. Das System trägt den Titel LMS Oxford Learn und lässt sich in drei Schichten unterteilen [AB17]: Die unterste Schicht besteht aus einer Reihe von Netzwerk-

komponenten und implementiert das Herunterladen bzw. Importieren von Multimedia-Inhalten sowie die Generierung von passenden Metadaten. Die mittlere Schicht organisiert die Inhalte in Datenbanken und macht sie den Nutzern durch Filterfunktionen zugänglich. Die oberste Schicht ist das UI, dies betont die Kollaboration von Nutzern in einer Forum-Umgebung [AB17]. Eine grob ähnliche Struktur unterliegt dem hier entworfenen System.

2.2 Interoperabilität von E-Learning-Systemen

Interoperabilität bezeichnet die Kompatibilität von Systemen bezüglich des Austausches und der Wiederverwendbarkeit von Daten und Inhalte. Diese Sektion trägt Forschung zu diesem, für diese Arbeit zentralen Thema zusammen.

Wiederverwendung von Inhalten Ainsworth et al. [AF06] zeigten bereits 2006, dass die Vorteile der Wiederverwendung von E-Learning-Inhalten enorm sind: Für eine Stunde Material reduzieren sie den Aufwand von 100 Stunden auf vier Stunden. Sie identifizierten vier Erfolgsfaktoren für die Wiederverwendung von Inhalten [AF06]:

1. Die Usability und Akzeptanz in der (oft nicht technisch versierten) Zielgruppe
2. Die Tatsächliche Qualität der Lernressourcen, d.h. der Lernerfolg sollte messbar sein
3. Die Technische Umsetzung der Interoperabilität, also die Nutzung von Standards und das Erfordern eines vertretbaren Aufwands für die Integration der Inhalte
4. Rechtliche Aspekte, insbesondere Open-Source-Lizenzen

SCORM, Tin Can API und xAPI Einer der ältesten Standards für den Austausch von E-Learning-Inhalten ist *SCORM* [Bak+17; Nie08; ADL09]. Er wurde von der Advanced Distribution Learning Initiative (ADL) entwickelt und besteht aus drei Komponenten [Nie08] :

1. Content Aggregation Model (CAM): Eine XML-basierte Spezifikation für die Identifikation und Aggregation von Inhalten in einem Manifest. Dieses bietet außerdem Informationen über Navigation, Kurstruktur und Metadaten.
2. Run-time Environment (RTE): Die Spezifikation der Anforderungen an ein LMS, diese beziehen sich auf Befehle, Interfaces und Datenmodelle für den Austausch zwischen Inhalten und Management-Komponenten sowie das Erfassen von Ergebnissen.
3. Sequencing and Navigation (SN): Eine Spezifikation der Darstellung der Inhalte, u.a. in Form von Aktivitäsbäumen, die die Berücksichtigung von Nutzer-Input ermöglichen.

Die *Tin Can API* is eine Weiterentwicklung von SCORM [ADL09] und fügt das Tracking und Speichern von Nutzern, deren Daten und Ergebnissen in mehreren Kontexten wie z.B. Mobile

Apps hinzu. Deren dritte Revision entspricht der *Experience API* (xAPI) [Nie08; Ini17]. Diese sieht das Speichern von Daten in LRS (Learning Record Stores) vor. Dies geschieht insbesondere via xAPI-Statements im Actor-Verb-Object-Format [Bak+17].

Learning Object Meta Data Bei Learning Object Meta Data (LOM) [IEE02] handelt es sich um einen IEE-Standard, der eine Menge von Daten vorgibt, die zusammen mit den eigentlichen Inhalten gespeichert werden [Nie08]. Das Konzept eines *Learning Objects* ist hier sehr breit definiert, es muss sich nicht unbedingt um digitale Objekte handeln und ein Objekt kann selbst aus mehreren Unter-Objekten bestehen. Die zu speichernden Metadaten lassen sich einer von neun Kategorien zuordnen [Nie08]:

1. *General*: Titel, Sprache etc.
2. *Lifecycle*: Entstehungszeit, Revisionsnummer etc.
3. *Meta-Metadaten*: Schema der Beschreibung selbst
4. *Technical*: Plattformanforderungen, Größe etc.
5. *Educational*: Altersgruppe, Kontext etc.
6. *Rights*: Kosten, Urheberrecht etc.
7. *Relation*: Verwandte Ressourcen
8. *Annotation*: Erfahrungswerte von Nutzern
9. *Classification*: Zweck, Kurzbeschreibungen etc.

IMS-Spezifikationen Das IMS Global Learning Consortium entwickelt eine Reihe von Standards für E-Learning, die Interoperabiltität sicherstellen sollen, die wichtigsten sind [Con21]:

- *LTI* - Learning Tools Interoperability [IMS19a]: Ein Standard für die Verbindung von E-Learning-Systemen und die nachfolgende Kommunikation zwischen diesen. Ein System tritt dabei als Provider und ein anderes als Consumer auf. Dieser Standard wird in dieser Arbeit intensiv verwendet und in Kapitel 3 näher erläutert.
- *CC* - Common Cartridge: Ein Standard für die XML-basierte Definition von E-Learning-Inhalten. Er gibt die Struktur und das Vokabular vor, um bestehende Kurse o.ä. für andere Systeme verständlich zu beschreiben [IMS15].
- *QTI* - Question and Test Interoperability: Eine Sepzifikation für Formate und Protokolle zur Einbindung von Tests in Learning Management Systeme [IMS20].
- *APIP* - Accessible Portable Item Protocol: Ein Datenmodell für den Austausch von Test-Items, das als eine Weiterentwicklung von QTI betrachtet werden kann.

Interoperabilität ohne Standards Then et al. [The+16] stellen ein Moodle-Plugin vor, das die Verwendung von Legacy-Infrastrukturen ohne LTI-Unterstützung analog zu mit LTI eingebundenen System erlaubt. Konkret wird die Assignment Software WA als LTI-Provider eingebunden mit dem Ziel Transparenz und somit Kompatibilität mit bereits existierenden Assignment Plugins herzustellen. Als theoretische Grundlagen und Analoga zu bestehenden Standards werden eigens entworfene Konzepte angewendet [The+16].

Karavirta et al. [KIK13] wenden einen serviceorientierten Ansatz auf das Interoperabilitätsproblem an. Es wird eine Applikation zur Übungsauswertung mit dem Titel *A+* entworfen. Diese kann via einer REST-API angesprochen werden. Um Interoperabilität herzustellen werden Übungen einer von vier Auswertungs-Kategorien zugeordnet [KIK13]:

1. Exercise Descriptions
2. Synchrone Auswertungen
3. Asynchrone Auswertungen
4. Statische Auswertungen

Die Aufgabe des verbundenen Systems ist es dann (nur noch) die Übungen auf eine der vier Kategorien abzubilden [KIK13].

Aleven et al [Ale+16] untersuchen die Einbettung von intelligenten Tutor-Programmen in verschiedene LMS. Diese können mit oder ohne Standards eingebunden werden. Die Tutoren sind alle mit den Cognitive Tutoring Authoring Tools (CTAT) [Ale+06] erstellt worden. Für die Integration in die LMS muss allerdings das Frontend dieser Software neu implementiert werden um die Erstellung von HTML-basierten Interfaces und die Nutzung des Browsers als Plattform zu ermöglichen. Die Integration der Tutoren kann dann via SCORM [ADL09], LTI oder via nativer Technologie der LMS stattfinden, z.B. durch die Open edX xBlock-Technologie [Ale+16]. In dieser Arbeit werden die Technologien LTI und xBlocks gemeinsam verwendet.

2.3 Web-Crawling im E-Learning-Kontext

Die meisten Crawling-Ansätze für E-Learning-Inhalte nutzen *Focused Web Crawler*, d.h. Systeme, die auf Basis einer Menge von Keywords nur relevante Inhalte entdecken. In der Formulierung von Chakrabarti et al. [CD99] findet eine Hyperlink-Discovery via Classifier und Distiller auf Basis der Suchterme statt. Der Crawler, der im Rahmen dieser Arbeit entworfen wurde folgt ähnlichen Grundsätzen.

Nicht-Ontologie-basierte Crawler Biletsky et al. [BWB09] entwerfen einen Focused Web Crawler, um große Anzahlen von E-Learning-Objekten mit minimalem User-Input herunterzu-

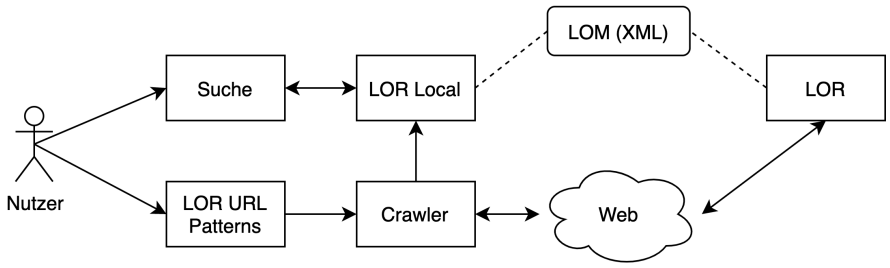


Abbildung 2.3: Übersicht über den Ablauf des LOM-Crawling, basierend auf [BWB09].

laden. Hierzu werden systematisch Websites durchsucht und deren Learning Object Metadaten (LOM) heruntergeladen. Die Architektur des Crawlers besteht aus einem ID Web-Crawler und einem LOM-Downloader. Der Ablauf des Crawlings ist in 2.3 dargestellt. Er ist wie folgt [BWB09]:

1. Zunächst erfolgt der Zugriff auf die Web-Learning Object Repositories (LORs).
2. Die Websites werden gemäß von LOR URL Patterns geparst.
3. Die LOM-Dateien werden in ein lokales Repository (LOR Local) heruntergeladen.
4. Die Dateien enthalten Referenzen auf die eigentlichen Inhalte und können von Services und Suchen verwendet werden.

Hijazi und Itzmazi [HI13] verwenden ein ähnliches System um nicht die Metadaten, sondern direkt die konkreten Inhalte herunterzuladen. Zhao et al. [Zha+16] entwerfen eine zweistufige Architektur für Deep Web Crawler mit dem Namen *SmartCrawler*. Die erste Stufe ist das *Site Locating*: Für eine Menge von Seed-Seiten fängt der Crawler an Links zu anderen Seiten in einer Datenbank zu sammeln und diesen zu folgen. Wenn der Anteil der unbesuchten Seiten in der Datenbank unter einen Schwellenwert fällt beginnt das *reverse searching*. Mit existierenden Suchmaschinen wird versucht *center pages* in den unbesuchten Seiten zu identifizieren [Zha+16]. Um die Ergebnisse zu verbessern werden weiter ein Site Ranker und ein Site Classifier verwendet. Bei der zweiten Stufe handelt es sich um das *In-Site Exploring*: Wenn eine relevante Seite in der ersten Stufe identifiziert wurde wird sie in der zweiten exploriert, eventuelle Formulare werden durch Form Classifier und Links durch Link Ranker systematisch exploriert [Zha+16].

Ontologie-basierte Crawler Ibrahim und Yang [IY19] untersuchen inwiefern Ontologien das fokussierte Crawling erleichtern können. Die zentrale Herausforderung eines Focused Crawler ist das Bestimmen der Relevanz einer Seite für das gegebene Thema - Ontologien können dieses Problem lösen. Zu Beginn des Crawling-Prozesses wird eine (nutzerkonstruierte)

Ontologie als Parameter übergeben. Entdeckte Dokumente werden dann mit Konzepten dieser annotiert und, falls sie ähnlich genug sind, dem Index hinzugefügt. Die Ähnlichkeitsberechnung findet mit einer adaptieren Version des TFIDF-Algorithmus statt [Ram+03], dieser ist in der folgenden Gleichung gegeben:

$$d_x = \frac{freq_{x,d}}{\max\limits_y freq_{x,y}} \log \frac{|D|}{n_x}$$

Sei D die Menge der mit Konzepten einer Ontologie O annotierten Dokumente. Jedes Dokument $d \in D$ ist repräsentiert als Vektor der Konzept-Gewichte. Für Konzept $x \in O$ und $d \in D$ ist d_x die Wichtigkeit von x in d. $\text{freq}_{x,d}$ ist die Anzahl der Vorkommen von x in d, n_x ist die Zahl der Dokumente, die von x annotiert sind. Das Ranking eines Dokumentes d ist dann gegeben via der Cosinus-Ähnlichkeit der Vektoren des Dokuments d und der Query q [IY19]. In dieser Arbeit werden die Konzepte des TFIDF-Algorithmus, der Ontologien und der Cosinus-Ähnlichkeit ebenfalls genutzt, jedoch nicht im Crawling-Kontext.

Iancu [Ian19] entwickelt einen Crawler für YouTube-Videos. YouTube ist eine der größten Informationsplattformen der Geschichte, die gezielte Suche von Videoinhalten ist aber inhärent schwer. Der hier verfolgte Ansatz ist das Extrahieren oder Generieren von Untertiteln und das anschließende Extrahieren der Entitäten einer Ontologie daraus. Dies erfolgt (wie bei dieser Arbeit auch) mit dem DBpedia Spotlight-Algorithmus. Die Ergebnisse werden dann in einer Graphdatenbank gespeichert. Das Ergebnis ist ein semantisches Netz. Konkret ist der Ablauf wie folgt [Ian19]:

1. Zunächst wird eine bestimmte Domäne gewählt, dann wird eine SPARQL-Anfrage bei DBpedia durchgeführt, diese identifiziert relevante Konzepte.
2. Dann findet eine Anfrage bei der YouTube Data API mit diesen Keywords statt.
3. Für die zurückgelieferten Videos werden Untertitel generiert oder selbst wieder angefragt.
4. Die Untertitel werden dann in einem CosmosDB-Index gespeichert.
5. Es werden ontologische Konzepte mit einem NER-Algorithmus extrahiert.
6. Nun ist eine Keyword-Suche auf den Ergebnissen möglich.

Premlatha und Geetha [PG11] entwickeln einen Web-Crawler für Lerninhalte, der Ontologien nutzt, um deren Relevanz abzuschätzen. Dies funktioniert in drei Phasen, dargestellt in Abbildung 2.4 [PG11].

- Phase 1: Hier werden auf auf der Basis von übergebenen Query-Konzepten Keywords generiert und expandiert, um eine Menge von Seed-Dokumenten zu bestimmen.

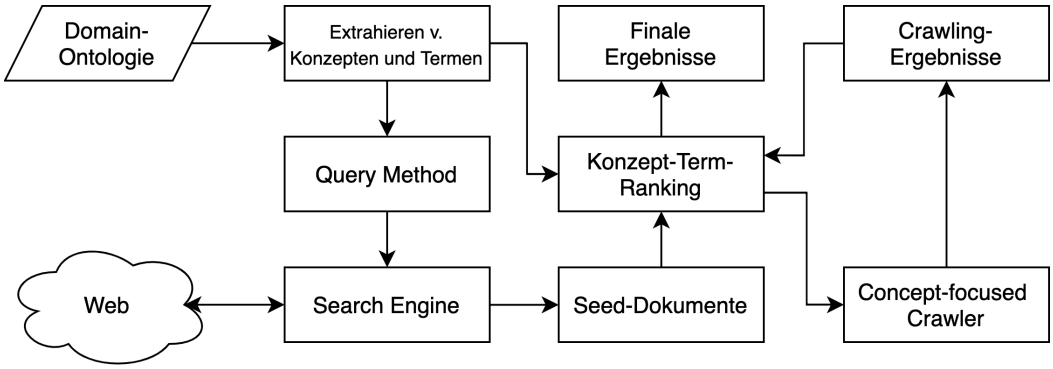


Abbildung 2.4: Übersicht über den Ablauf des Focused Crawlers [PG11].

- Phase 2: Hier werden die identifizierten Dokumente bezüglich der Konzepte der Ontologie bewertet sowie Redundanzen eliminiert.
- Phase 3: Hier findet ein Focused Crawling statt, das die Menge der Dokumente erweitert und ein erneutes Ranking initiiert.

Vertikale Suche Kolli und Pavani [PS17] entwickeln die Crawling-Methode der vertikalen Suche. Es gibt Tradeoffs im Indizieren von Datenmengen für Suchmaschinen: Je mehr Inhalte indiziert werden desto öfter werden bereits gescannte Seiten aktualisiert und desto schwieriger ist es relevante Antworten auf eine Suchanfrage zu liefern. Themenspezifische Suchmaschinen sind von diesen Problemen weniger betroffen, da sie Focused Crawler verwenden [PS17]. Ein Beispiel für den Unterschied ist die Google-Suche im Vergleicht zu Google Scholar. Auf der anderen Seite ist hier aber der „rich get richer"-Effekt von PageRank stärker und der Rank dominiert andere Relevanz-Kriterien [PS17]. Die vorgestellte Lösung besteht darin Ranks für alle Seiten zu berechnen und anschließend für Seiten mit einem Rank $t \in [t_{min}, t_{max}]$ die Ähnlichkeit zur Query zu berechnen. Nun werden nur Seiten durchsucht deren Ähnlichkeit zu den bereits gewählten URLs größer als ein Schwellenwert ist [PS17].

All diese Ansätze haben gemeinsam, dass eine domänenspezifische Ontologie existiert oder als Parameter übergeben wird, das ist in dieser Arbeit nicht der Fall.

2.4 Recommender-Systeme für E-Learning

Diese Sektion gibt eine Übersicht über die in der Literatur verwendeten Ansätze E-Learning-Inhalte vorzuschlagen, Ideen dieser werden auch in dem hier entwickelten Recommender-System wiederverwendet. Es sind jedoch anwendungsfallbedingt nicht alle umsetzbar.

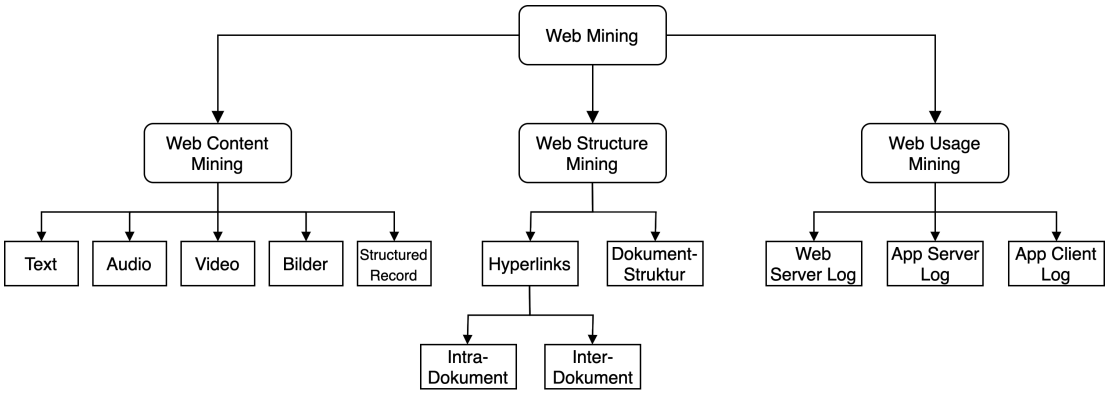


Abbildung 2.5: Übersicht über die verschiedenen Aspekte des Web Mining, modifiziert aus [UMN07]. Die drei Hauptkategorien sind die Untersuchung von Inhalten, der Struktur und der Benutzung von Webseiten.

Web Mining Umadevi et al. [UMN07] beschreiben die in Abbildung 2.5dargestellten Aspekte des Web-Mining und wie diese für E-Learning-Anwendungen verwendet werden können. Für Suchmaschinen und Content Discovery ist Content Mining essentiell. Soll dies automatisch geschehen (d.h. in einem Web-Crawler) muss allerdings auch Structure Mining zum Einsatz kommen. Für Recommender-Systeme kann insbesondere Usage Mining verwendet werden, z.B. durch den Aufbau von Profilen für anschließendes Collaborative Filtering [Sch+07]. Auf den extrahierten Daten können anschließend folgende Techniken verwendet werden [UMN07]:

1. Klassifikation
2. Clustering
3. Vorhersagen
4. Association Rules

Mögliche konkrete Implementierungen sind Neuronale Netze, Nearest Neighbor-Classifiers und Entscheidungsbäume, wie von Li und Zhang [LZ10] vorgestellt. Kumar und Pal [KP11] nutzen den ID3-Decision Tree um Lernende zu erkennen, die Hilfe benötigen. Dazu extrahieren die Autoren Variablen wie Seminarnoten, Noten von schriftlichen Prüfungen, Noten des vergangenen Semesters etc., um zukünftige Ergebnisse vorherzusagen.

Association Rules Zaiane [Zai02] nutzt das oben erwähnte Usage Mining um einen Agenten für inhaltliche Empfehlungen für E-Learning-Inhalte zu entwerfen. Dabei werden ür jeden Nutzer die Logs untersucht, die dort verzeichneten Aktionen gruppiert und somit Item Sets konstruiert. Diese können wiederum nun als Grundlage für einen Association-Rule-Algorithmus [AIS93] dienen, der Assoziationen zwischen Inhalten erkennen kann. Diese müssen allerdings

in der Regel noch weiter gefiltert werden, beispielsweise durch Gewichte - direkt erreichbare Inhalte sind keinen nützlichen Vorschläge, weit entfernte Inhalte, die trotzdem häufig zusammen auftreten sind dagegen sehr gute Vorschläge [Zai02]. Die Definition einer geeigneten Distanzmetrik ist selbst aber wieder eine Herausforderung. Lu [Luo4] kombiniert die Idee der Association Rules mit einem Framework für die Analyse der Bedürfnisse von Nutzern, um die Menge der vorgeschlagenen Inhalte zu verfeinern. Hier werden Nutzer und Materialien als Vektoren betrachtet und *Fuzzy Matching* durchgeführt.

Collaborative Filtering Tang und Mccalla [TM03] adaptieren klassisches Collaborative Filtering auf das Vorschlagen von Research Papers basierend auf ihren Nutzerbewertungen. Die Papers haben Features in Form von Tags, es gibt zwei Kategorien:

1. Content Tags: Titel, Kategorie in Form von Keywords, Datum, Publikation etc.
2. Technical Tags: Zielpublikum, Stil, Nutzen

Ebenso gibt es Mandatory Readings, die stets empfohlen werden. Auf der Nutzerseite wird sogenanntes Focused Collaborative Filtering durchgeführt: In einem ersten Schritt werden Nutzer geclustert und nur innerhalb dieser Cluster wird der CF-Algorithmus durchgeführt. Ein weiterer Teil des Systems ist die Intelligent Paper Maintenance für das Aktualisieren des Repositories, das Aktualisieren von Tags und die Garbage Collection schlechter Papers [TM03].

Kombination von Techniken Khribi und Jemni [KJN08; JKN07] kombinieren die obigen Techniken des Web Mining, Association Rules und Collaborative Filtering in einem System. Der Ansatz ist in zwei Phasen unterteilt, die Modeling Phase (offline) und die Recommendation Phase (online). In der Modeling Phase werden basierend auf Log-Files Sessions identifiziert und dann geclustert. Diese Cluster werden Profile genannt. Sei $U = \{u_1, \ldots, u_n\}$ die Menge aller URLs. Nach dem Usage und Content Mining erhält man die Sessions $S = \{s_1, \ldots, s_m\}$ mit $s_i \subseteq U$. Nun findet Association Rule Mining statt. Ziel ist das Finden von häufig gruppierten Inhalten in den Sessions [KJN08; JKN07]. Es werden ebenfalls alle Inhalte in einem Invertierten Index gespeichert, als Teil des sogenannten *Content Profiling*. Nun kann die Recommendation Phase stattfinden. Diese beginnt mit dem Extrahieren der aktiven Session, einem Sliding Window der letzten W relevanten URLs. Auf dieser Basis wird Content Based Filtering und Collaborative Filtering durchgeführt sowie Association Rule Mining und das Extrahieren der Such-Terme der URLs und deren Eingabe in eine Suchmaschine [KJN08; JKN07]. Es handelt sich also um einen hybriden Ansatz.

Tarus et al. [TNY17; TNM18] nutzen eine Learner Model-Ontologie, eine Resource Model-Ontologie und Association Rule Mining für das Vorschlagen von Inhalten.

Rahman und Abdullah [RA18] schlagen eine System vor, dass Google-Suchen und LMS-interne Suchen kombiniert. Nach einer Authentifizierung beim LMS wird eine Suchmaske angezeigt, hier eingegebene Suchen werden an Google und das LMS weitergeleitet und die Ergebnisse vereint angezeigt, die Reihenfolge der internen Ergebnisse ist dabei allerdings nutzerabhängig (durch unterschiedliche Prioritäten von URLs). Die Prioritäten werden dabei durch den Prozess des *Student Profiling* berechnet [RA18]:

1. Via Usage Mining werden alle Profile als Anfänger, Fortgeschritten oder Experte klassifiziert: Ein Crawler durchsucht alle 72 Stunden sämtliche Informationen über die Nutzer, Logs über besuchte Inhalte, Suchanfragen und Lernerfolge.
2. Die Module des Academic Record Analyzer und Behavioral Activity Analyzer liefern zusätzliche Informationen.
3. Nun findet die Klassifizierung der Nutzer-Profile durch Entscheidungsbäume statt.

Danach findet eine Re-Ranking der Inhalte durch das gruppenbasierte Zuweisen von Prioritäten statt, die URL-Prioritäten ergeben sich auf Basis des ursprünglichen Rangs und des oben berechneten Gruppen-Scores [RA18].

Kontext Gallego et al. [Gal+13] entwerfen ein kontextbasiertes System für proaktive inhaltliche Vorschläge für ein Authoring Tool. Der Kontext besteht aus zwei Teilen: dem User Context (z.B. die aktuelle Aktivität) und dem Educational Context (Thema, Sprache, Zielgruppe des bearbeiteten Materials). Der Kontext muss stets bekannt sein, d.h. initial muss er manuell eingegeben werden und wird nachfolgend auf Basis der Metadaten der verwendeten Lernobjekte verfeinert. Es wird kontinuierlich ein *Situation Assessment* auf Basis der Nutzeraktivitäten durchgeführt [Gal+13]. Dessen Ergebnis ist ein Score $S_1 \in [0,1]$. Falls dieser über einem Schwellenwert liegt wird dem Nutzer eine Empfehlung angezeigt, beispielsweise ist dieser Score höher wenn gerade eine Pause vorliegt und niedriger wenn gerade aktiv ein Video angeschaut wird. Gleichzeitig zum Situation Assessment wird auch Resource Assessment durchgeführt. Basierend auf den verwendeten Ressourcen wird für jedes andere Objekt ein Score $S_2 \in [0,1]$ berechnet und nur Elemente mit ausreichend hohen Scores kommen für eine Empfehlung in Frage [Gal+13].

Similarity Degree Faqihi et al. [FDA19] berechnen inhaltliche Vorschläge auf Basis der Ähnlichkeit ihrer Metadaten zueinander. Hierzu definieren sie zunächst eine Ontologie der Struktur von E-Learning-Ressourcen. Gemäß dieser Ontologie besteht eine Ressource aus Stichworten (Tags), einem Ziel, einer Granularität, einem Medienformat etc. Empfehlung werden

dann basierend auf der Ähnlichkeit dieser Metadaten berechnet. Hierzu können existierende Metriken wie die Cosinus-Distanz verwendet werden [FDA19].

Die Vorraussetzungen für die oben genannten Recommender-Systeme sind wie erwähnt in dieser Arbeit nicht gegeben, aber Ideen wie die des Re-Rankings oder des Similarity Degrees werden auch hier angewandt.

2.5 Information Retrieval von E-Learning-Inhalten

Guemmat et al. [ET18; EO19] identifizieren eine Reihe von Herausforderungen von Suchmaschinen im Kontext des E-Learning. Diese sind:

- Fehlende Interoperabilität: Es ist nicht möglich ohne Weiteres E-Learning-Inhalte zwischen Systemen auszutauschen oder direkt auf diese zu verlinken, es gibt keine ad-hoc nutzbaren Standardformate.
- Manuelle Metadaten: Für die Suche (z.B. als Filter) nutzbare Metadaten wie der Typ oder die Zielgruppe sind nicht generierbar sondern müssen für alle Ressourcen manuell erstellt und gepflegt werden.
- Versteckte Lernobjekte: In der Regel kann nicht auf Subobjekte wie Seiten o.ä. in einem Modul der LMS-Organisationshierarchie verlinkt werden, diese können also nicht oder nicht in Isolation als Suchergebnisse verwendet werden.
- Multimedia: Für Filme, Spiele etc. ist eine Volltextsuche nicht geeignet, der Mehrwert von Lösungen mittels generierten Annotationen ist im Allgemeinen sehr beschränkt.

Um Probleme dieser Art zu lösen, wird die in Abbildung 2.6 dargestellte Suchmaschine entworfen, die auf der Nutzung der Semantic Web-Technologie des RDF (Resource Description Framework) basiert [ET18; EO19]. Für alle Anfragen an den Index wird die SPARQL-Anfragensprache verwendet, dies erfordert auf der einen Seite die Konversion von natürlichsprachigen Anfragen in ein RDF-Format und auf der anderen die Annotation aller indizierten Lernobjekte mit Konzepten einer RDF-kompatiblen Ontologie. Beide sind technisch herausfordernd, werden aber von den Autoren nicht weiter beschrieben. Bei der hier vorliegenden Arbeit gibt es ein ähnliches Problem zu lösen.

Alharbi et al. [AHH12] entwickeln eine domänenspezifische Suchmaschine für Informatik-Lernmaterialien. Dazu wird eine Erweiterung des IEE LOM-Standards [IEE02] entworfen. Diese basiert auf dem ACM Computing Classification System, eine hierarchische Taxonomie für Themen der Informatik.[1] Alle suchbaren Elemente werden mit diesem Metadaten-Standard

1 https://dl.acm.org/ccs

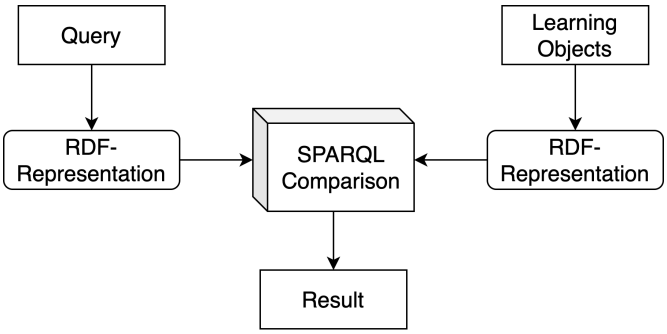


Abbildung 2.6: Die Architektur der hybriden Suchmaschine, modifiziert nach [EO19]. Inhalte und Suchen werden stets auf eine RDF-Repräsentation abgebildet.

annotiert. Der eigentliche Information Retrieval-Vorgang geschieht nun via Text-Matching auf dem kontrollierten Vokabular der Annotationen [AHH12].

Nasraoui und Zhuhadar [NZ10] diskutieren eine Methode, um die Performance semantischer E-Learning-Suchmaschinen zu verbessern. Sie besteht aus den folgenden vier Phasen:

1. Aufbau einer semantischen E-Learning-Domäne: Die Domäne ist eine hierarchische Baum-Struktur, die aus Konzepten mit eventuellen Sub-Konzepten besteht.
2. Generieren von Nutzerprofilen in der Form von Ontologien: Auf Basis von demographischen Informationen etc. können für jeden Nutzer dessen Interessen festgestellt und gespeichert werden.
3. Clustering von Dokumenten: In diesem Schritt werden die Dokumente jedes Clusters mit Sub-Subkonzepten annotiert.
4. Re-Ranking von Sucherergebnissen gemäß der Nutzerprofile: Je mehr Konzepte ein Ergebnis und das Nutzerprofil (z.B. in Form von Interessen) gemeinsam haben desto relevanter ist es.

Arai und Tolle [AT11] entwerfen drei Methoden, um die Effizienz von Suchmaschinen für E-Learning zu erhöhen, insbesondere relevant ist dies für mobile Geräte. Bei den Methoden handelt es sich konkret um die folgenden:

- Das Filtern nach Dateitypen: In der Regel kann der Nutzer nur wenige Dateitypen verwenden, das entsprechende Verkleinern des Suchraumes auf nur bestimmte Typen ist naheliegend. Da allerdings nur wenige Nutzer diese Methode selbst anwenden sollte sie im User-Interface prominent dargestellt, wenn nicht gar automatisch aktiviert werden.
- Keyword-Insertion: Die meisten Keywords, die in eine Suchmaschine eingegeben werden haben unterschiedliche Bedeutungen in unterschiedlichen Domänen. Um damit

umgehen zu können werden für die Worte eines festen Vokabulars inhaltliche Beziehungen vorberechnet und, falls mehrere Stichworte gegeben sind, die Schnittmenge dieser Beziehungen als Suchraum verwendet - zum Beispiel ist die Domäne „Programming“ für eine Suche nach „Java“ relevant, für die Suche „Java Island“ aber nicht, da sie nicht in der Schnittmenge der beiden Keywords liegt.

- Previews statt Downloads: Auf technischer Seite ist es sinnvoll für die Präsentation von Suchergebnissen nicht die ganze Datei zu verwenden, sondern eine leichtgewichtige Präsentation derselben.

Taibi et al. [TGS05] stellen zwei Methoden vor, um das semantische Suchen von E-Learning-Inhalten zu ermöglichen: die Static Ontology Based Search (SOBS) und die Dynamic Ontology Based Search (DOBS). In beiden Fällen wird ein Web-Crawler verwendet, um Dokumente zu entdecken und in einen Index einzutragen. Diese werden danach in einem Information Extraction-Schritt auf ihre Semantik analysiert und mit den jeweils relevanten Konzepten bzw. Entitäten einer Ontologie annotiert. Im Rahmen der SOBS findet die Annotation direkt im Anschluss an den Index-Vorgang statt, es muss also eine statische Ontologie vorliegen. Bei der DOBS findet die Annotation erst on-demand statt, es wird dabei keine eingebaute Ontologie verwendet sondern eine vom Nutzer übergebene. Die Annotation bzw. die Konstruktion des Suchraums kann dann ggf. auf nur einer Teilmenge des gesamten Corpus stattfinden. Die Ontologie kann ebenfalls als Teil eines Reasoning-Moduls verwendet werden, das Beziehungen zwischen Dokumenten ableiten kann [TGS05].

3 Grundlagen

Dieses Kapitel fasst die wichtigsten konzeptionellen Grundlagen zusammen, die für den Entwurf des Assistenzsystems relevant sind. Zunächst wird das Konzept des E-Learning in Sektion 3.1 genauer erklärt, dann Assistenzsysteme in Sektion 3.2, die Interoperabilität von Systemen in Sektion 3.3, Web-Crawling in Sektion 3.4, Information Retrieval in Sektion 3.5, relevante Webtechnologien in Sektion 3.6 und am Ende Natural Language Processing in Sektion 3.7.

3.1 E-Learning: Inhalte und Verwaltung

Das in dieser Arbeit entworfene Assistenzsystem implementiert eine Suchmaschine für E-Learning-Inhalte und interagiert extensiv mit Learning Management Systemen, beide Konzepte werden nachfolgend genauer erläutert.

E-Learning In seiner allgemeinsten Form beschreibt der Begriff des E-Learning (nur) das Präsentieren von Lerninhalten in digitaler Form, in der Regel als Teil von Webseiten. Wie später auch in Kapitel 4 thematisiert wird, hat der Begriff des Lerninhaltes keine strenge Bedeutung - nahezu alle Inhalte, egal in welcher Domäne oder Form, können als Lerninhalte dienen. Downes [Dow05] konkretisiert und definiert E-Learning als das organisierte Anbieten und Präsentieren von Learning Objects, d.h. zusammenhängende Mengen von Inhalten, in der Regel in Form von Online-Kursen. Deren Zusammenhang und Auslieferung kann, muss aber nicht, standardisiert erfolgen [Dow05]. Bei den Inhalten kann es sich um eine breitgefächerte Menge von Typen handeln: Texte, Videos, Quizzes, Aufgabenstellungen, Interaktive Elemente, Spiele, soziale Aktivitäten, Offline-Aktivitäten etc. Auch Kombinationen von Typen sind möglich.

Learning Management Systeme E-Learning-Inhalte sind meistens in Learning Management Systemen (LMS) organisiert, dabei handelt es sich um Server-Anwendungen, die das Erstellen und Verwalten von Inhalten erlauben - hier sind die Kurse zuzuordnen. Die Interaktion der Nutzer mit den Inhalten erfolgt in Form eines Web-Frontend, dass auch Teil des LMS ist. In den meisten Fällen existiert auch ein Account-Konzept, dass die Authentifizierung und Autorisierung von Nutzern durch das kontrollierte Einschreiben in Kurse oder verschiedene

Berechtigungsklassen ermöglicht. Klassische Beispiele sind Lehrende, die Kurse gestalten und Lernende, die diese bearbeiten. Diese Accounts können auch an die Accounts der Institution gekoppelt sein, die das LMS betreibt. In Verbindung mit dem Account-Konzept gibt es auch zahlreiche andere Features, die man mit klassischen Kursen assoziiert, z.B. Foren, Kalender, Gruppeneinteilungen, Hausaufgaben und deren Bewertung. In dieser Arbeit werden drei LMS betrachtet, die nachfolgend genauer erklärt werden. Die angebotenen Features sind bei allen drei ungefähr äquivalent. Nach einem regulären Login werden Nutzer zunächst auf ein Dashboard weitergeleitet, hier werden alle Kurse angezeigt, für die der Nutzer eingeschrieben ist. Je nach Betriebsmodus des LMS oder der Organisation desselben ist es hier auch möglich nach anderen Kursen zu suchen und sich für diese entweder direkt einzuschreiben oder eine Mitgliedschaft zu beantragen. Oft wird hierzu ein Passwort benötigt oder eine manuelle Einschreibung ist gar nicht möglich und Nutzer müssen von Kursveranstaltern separat hinzugefügt werden. Der Zugriff auf die eigentlichen Inhalte erfolgt nach der Einschreibung innerhalb von Kursen. In den Abbildungen 3.1, 3.3 und 3.5 sind Screenshots von Kursen für Moodle [AZ08], Ilias [Bed+13] und Open edX dargestellt. Wie dort zu erkennen ist, wird stets ein großer Wert auf das grafische User Interface gelegt, unter anderem durch visuelle Strukturierung und Icons. Die unterliegende Technologie dieser Interfaces ist HTML. Dieses HTML wird im Rahmen dieser Arbeit von einem Web-Crawler durchsucht, LTI [IMS19a] abstrahiert dabei die Kurs-Einschreibung. Die (wichtigsten) LMS-Features, die so präsentiert werden sind:

- Der Zugriff auf die eigentlichen Lerninhalte, dazu zählen:
  - normale Seiten, die Textabschnitte mit Bildern oder Videos kombinieren
  - strukturierte Sammlungen dieser Seiten in Büchern, Sequenzen oder Wikis
  - Aufgaben, die die Abgabe von Ressourcen wie Essays erfordern
  - Quizzes, die durch die Angabe von (kürzeren) vorgegebenen oder frei zu formulierenden Antworten bearbeit werden
  - Foren oder E-Mail-Klienten, die die Kommunikation der Kursteilnehmer ermöglichen
  - Eingebundene Dateien, die heruntergeladen werden können
- Die organisatorischen Aspekte wie:
  - Die (hervorgehobene) Darstellung von Ankündigungen
  - Die Darstellung von Kalendern und Terminen
  - Die Darstellung von Korrekturen und Bewertungen
- Die technischen Aspekte wie:
  - Die Navigation in Kursen durch Legenden oder Timelines

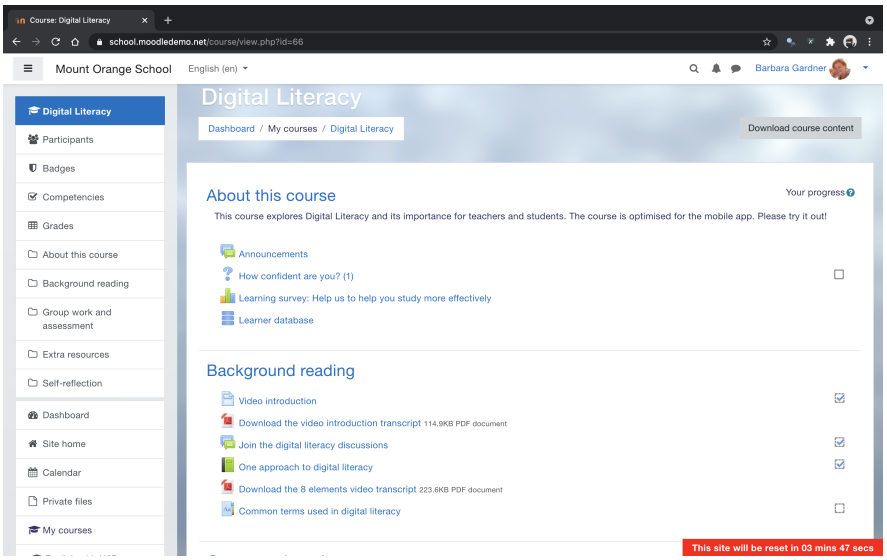


Abbildung 3.1: Screenshot eines Moodle-Kurses aus [Moo21a].

- Die Navigation zwischen Kursen
- Der Zugriff auf Optionsmenüs

Im Rahmen des Durchsuchen dieser Kurse mithilfe des Web-Crawlers gilt es nun relevante Inhalte zu identifizieren und dabei Seiteneffekte und unnötigen Overhead zu vermeiden. Nachfolgend werden die hierzu relevanten technischen bzw. softwarearchitekturellen Aspekte der drei betrachteten LMS dargestellt.

Wie der Entwickler-Dokumentation [Moo21b] zu entnehmen ist, besteht Moodle aus einem stets benötigten Core-System und zugehörigen Subsystemen sowie einer Menge optionaler Plugins verschiedener Komplexität, dargestellt in Abbildung 3.2. Das Core-System implementiert Aspekte wie die Kursstruktur und User-Enrolments, Plugins stellen die Lerninhalte, Themes und das in dieser Arbeit besonders relevante LTI-Enrolment bereit. Der zu entwerfende Crawler muss also die Navigation im Core-System ermöglichen und mit einer (ausgewählten) Menge von Plugins umgehen können. Die Architektur von Ilias ist in Abbildung 3.4 dargestellt und ist (bis auf die verwendete Terminologie) durchaus vergleichbar. Gemäß dem Ilias Development Guide [eV21b] besteht Ilias aus einer Server-Applikation, die die notwendigen Technologien wie Apache oder eine PHP-Installation bereitstellt. Die Funktionalität des LMS wird durch Module und Services implementiert. Module sind mit den Moodle-Plugins vergleichbar, sie stellen einzelne Ressourcen wie Quizzes oder Wikis dar. Services sind für mehrere Module relevant und ermöglichen beispielsweise das User-Enrolment und den hierfür nötigen Zugriff auf die Datenbank. Klassen beider Kategorien können wiederum selbst entweder dem

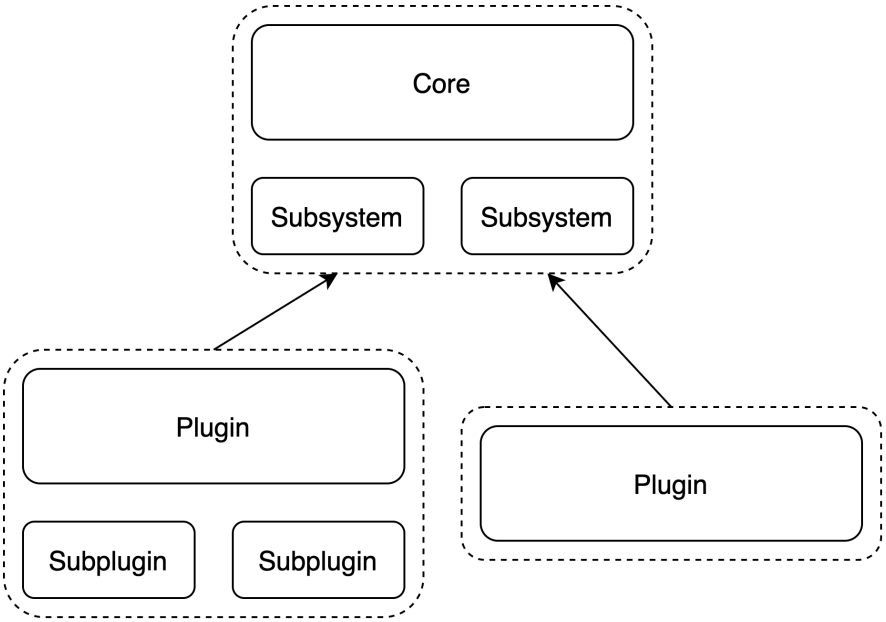


Abbildung 3.2: Die Architektur von Moodle, modifiziert von [Moo21b].

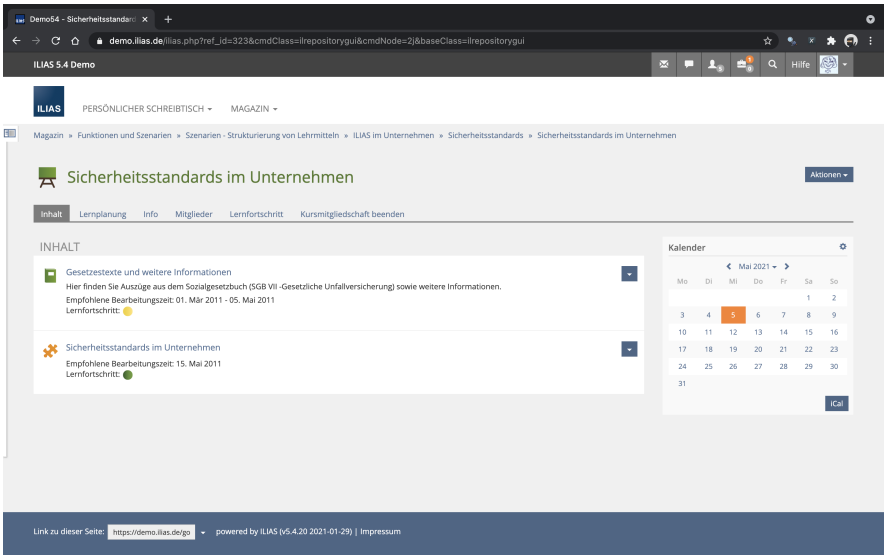


Abbildung 3.3: Screenshot eines Ilias-Kurses aus [eV21a].

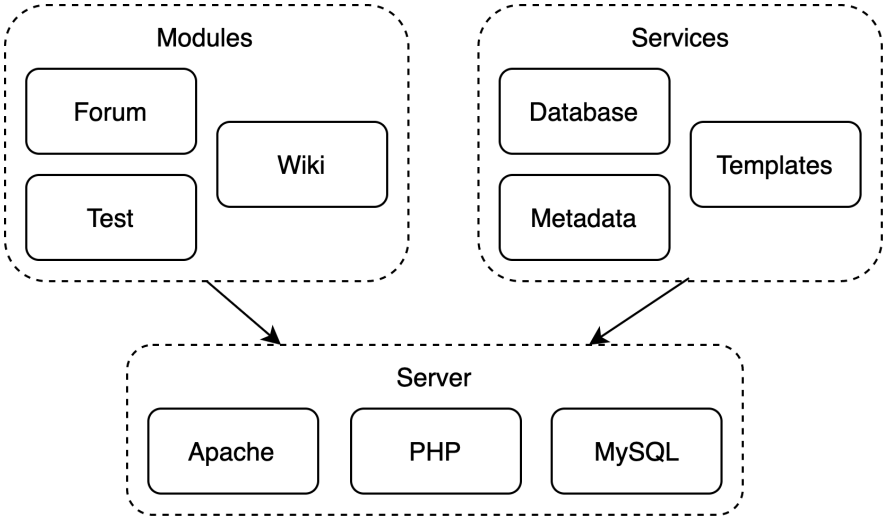


Abbildung 3.4: Die Architektur von Ilias, modifiziert aus [eV21b].

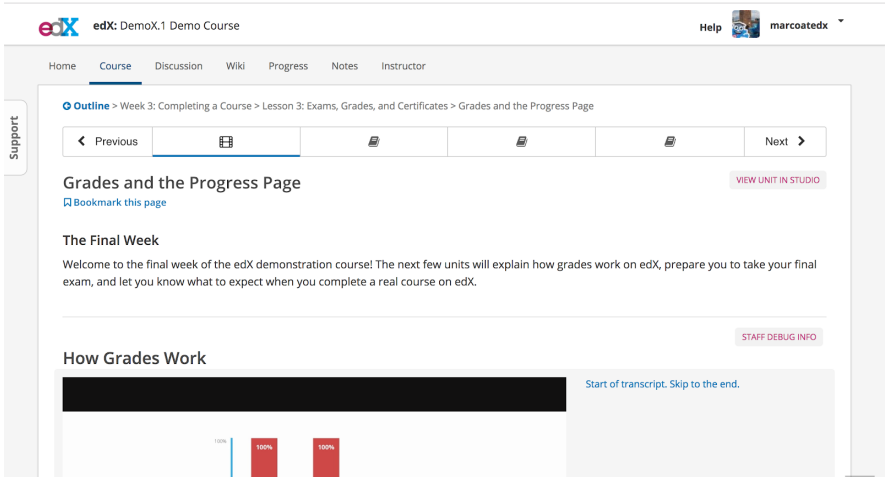


Abbildung 3.5: Screenshot eines Open edX-Kurs [Mor17].

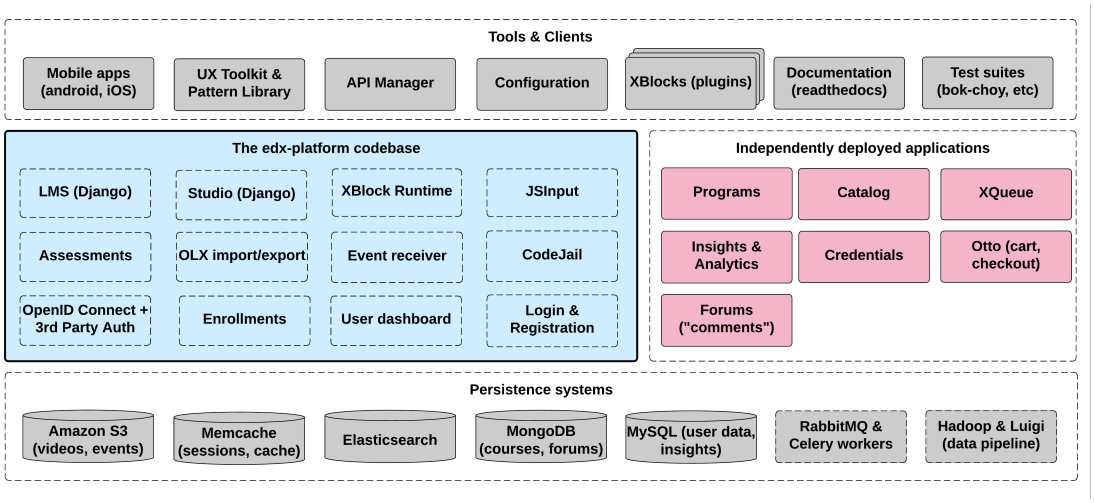


Abbildung 3.6: Übersicht über die Architektur von Open edX [Inc21].

UI-Layer oder dem App-Layer zugeordnet werden [eV21b]. Ein Crawler muss also folglich in den UI-Klassen der Module navigieren können und die Session via einer App-Klasse der Services eröffnen. Die Architektur von Open edX ist in Abbildung 3.6 aus dem Open edX Devloper's Guide [Inc21] dargestellt. An dieser Stelle sollte auch festgehalten werden, dass edX in der Regel die Organisation bezeichnet und Open edX die (open source) Plattform dahinter, oft werden die Begriffe jedoch auch synonym verwendet. Diese Arbeit verwendet stets Open edX. Die Architektur besteht aus vier Subsystemen: dem Persistenzsystem, das eine Reihe von Datenbanktechnologien enthält, Applikationen mit Deployment, das unabhängig von der konkreten Open edX Installation ist, einer Menge von Tools und Clients, und der eigentlichen Plattform. Für den zu entwerfenden Crawler sind die xBlock-Komponenten von essenzieller Bedeutung, diese werden als Tool indiziert und können (nach entsprechender Konfiguration) direkt per LTI-Launch geöffnet werden.

3.2 Assistenzsysteme und Autorenwerkzeuge

Wie Wandke et al. [WW03] anmerken, ist die Bedeutung des Begriffes des Assistenzsystems in der Regel vom Anwendungskontext abhängig und stark mit dem Konzept der Usability bzw. Nutzbarkeit von Systemen verbunden. Eine mögliche, von Wandke et al. [WW03] abgeleitete Definition ist die folgende: Assistenz bezeichnet die Erleichterung des Zugangs eines Nutzers zu der Funktionalität eines technischen, interaktiven Systems. Sie ist eine Brücke zwischen den Zielen und Fähigkeiten eines Nutzers und denen eines Systems. Beispiele für Assistenten und deren Anwendungsfelder sind Spurhaltungssysteme und Bremsassistenzsysteme in

PKWs, Hard- und Software in Smart Homes oder Asssitive Technology für Menschen mit Behinderung [WW03]s. Im hier relevanten Kontext sind aber vor allem grafische Editoren und Suchmaschinen zu nennen. Konkret können Assistenten in die folgenden Kategorien eingeteilt werden [WW03]:

- Motivation und Zielbildung: Assistenten für Orientierung und Warnungen
- Informationsaufnahme: Assistenten für die Anzeige, Betonung und Wiederholung von Informationen
- Informationsintegration: Assistenten für die Präsentation und Erklärung von Informationen
- Handlungsentschluss: Assistenten für das gezielte Vorschlagen von Handlungsmöglichkeiten
- Ausführen der Handlung: Assistenten für Eingaben und Abkürzungen
- Verarbeitung von Rückmeldungen

Assistenzsysteme können überdies gemäß ihrer Anpassbarkeit klassifiziert werden [WW03]:

- Konstante Assistenz: Systeme, die sich nicht an Änderungen des Kontexts anpassen. Sie sind transparent und konsistent, aber inflexibel.
- Anwenderspezifische Assistenz: Systeme, die vor ihrer Nutzung bzw. während der Entwicklung anpassbar sind, aber nicht während ihrer Nutzung.
- Adaptierbare Assistenz: Der Nutzer kann das System (bis zu einem bestimmten Grad) an seine Bedürfnisse anpassen, z.B. durch das selbstständige Setzen von Parametern.
- Adaptive Assistenz: Das System passt sich (semi-)automatisch an den Nutzer an durch die Auswertung von Kontextinformationen, insb. im Rahmen von ML-Ansätzen.

Zudem kann zwischen aktiver und passiver Assistenz unterschieden werden [WW03]. Im ersteren Fall kann das System proaktive Hilfestellungen geben, wenn eine Analyse ergibt, dass diese nützlich sein könnten, z.B. wenn eine bestimmte Komponenten zum ersten Mal oder für eine lange Zeit aufgerufen wurde. Passive Assistenz findet erst auf expliziten Wunsch des Nutzers statt, die Mächtigkeit der Ansätze muss sich allerdings nicht unterscheiden.

Bevan et al. [BKM91] analysieren die ISO-Definition von Software-Usability: Eine Menge von Attributen einer Software, die Einfluss auf den Aufwand ihrer Nutzung und die individuelle Einschätzung der Nutzung haben. Die Nutzbarkeit einer Software hat einen erheblichen Einfluss auf ihren Erfolg. Sie ist von den in Abbildung 3.7 dargestellten Faktoren beeinflusst, insbesondere relevant sind die Attribute des Produkts, die Aufgabe, die der Nutzer lösen möchte sowie die Kontexte der Umgebung und der Organisation [BKM91].

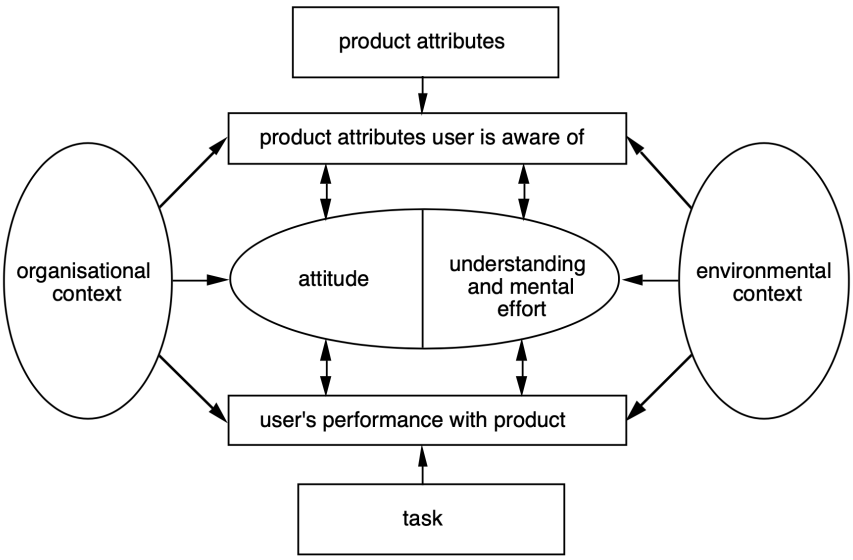


Abbildung 3.7: Die Aspekte, die die Usability eines Systems beeinflussen [BKM91].

Bei Autorenwerkzeugen handelt es sich um einen Spezialfall von Assistenzsystemen, die die Erstellung von Inhalten ermöglichen bzw. diese erleichtern. Der Begriff des Inhalts ist hier bewusst breit, er kann sich auf natürlichsprachige Texte, Code oder interaktive Medien beziehen. Im Folgenden werden (nur) Authoring Tools für E-Learning-Inhalte betrachtet. Für diese geben Khademi et al. [KHK15] eine genauere Definition an: Autorenwerkzeuge sind Programme, die es erlauben die Beziehungen und Reihenfolge von Objekten wie Textabschnitten oder Bilder zu definieren und so E-Lerning-Inhalte zu erstellen. Technisch ist dies durch eine Mischung aus Hypertext und grafischen Interfaces realisiert. Raabe et al [RCV15] definieren eine Taxonomie von E-Learning-Authoring Tools. Diese besteht aus den folgenden drei Kategorien und assoziierten Subkategorien:

- Content Development Tools: Tool Suites, Content Editing, Animation, Presentation Tools
- Multimedia Tools: Graphic, Video, Audio Tools
- Auxiliary Tools: Conversion, Communication, Evaluation, Gaming, Cognitive, Content Management, Collaborative Tools

Gordillo et al. [GBQ17] definieren zudem eine Reihe von Charakteristiken, um Autorenwerkzeuge beschreiben und bewerten zu können. Diese sind:

- Die Unterstützung verschiedener Plattformen
- Die Unterstützung verschiedener Dateiformate

- Die Konfigurierbarkeit erstellter Inhalte
- Den Automatisierungsgrad
- Die Bedienbarkeit
- Die Konformität zu Standards
- Die Kosten und Lizenz

3.3 Interoperabilität durch LTI und CLM

Diese Sektion erläutert die Interoperabilitätstechnologien, die grundlegend für die Funktionsweise des hier entworfenen Systems sind und beschreibt zudem deren weiteren Kontext.

Der Begriff der Interoperabilität Grosche und Wunder [GW09] definieren mehrere Ebenen der Interoperabilität von Systemen genereller Natur:

1. Fehlende Interoperabilität: Systeme sind nicht physisch verbunden und Informationsaustausch erfordert stets menschlichen Aufwand wie das manuelle Kopieren von Daten oder deren manuelle Modifikation.
2. Dateninteroperabilität: Es besteht eine physische Verbindung zwischen Systemen und Daten können empfangen werden, deren Verarbeitung ist allerdings nicht gefordert.
3. Syntaktische Interoperabilität: Empfangene Daten können verarbeitet werden, d.h. dass beispielsweise das Dateiformat vorgegeben und unterstützt ist.
4. Semantische Interoperabilität: Die Interpretation der gesendeten Daten von Sender und Empfänger ist gleich.
5. Pragmatische Interoperabilität: Nicht nur die Interpretation ist gleich, sondern auch die Vorhersage der Folgen der Information. Die Systeme können also als Module eines übergeordneten Systems gesehen werden.

Die im Folgenden verwendeten Lernstandards erzielen mindestens syntaktische Interoperabilität. Ob Stufe 4 oder 5 erreicht werden ist im Allgemeinen nicht klar. Im spezielleren Kontext von Informationssystemen definieren Park und Ram [PR04] Interoperabilität als die Erstellung einer semantisch kompatiblen Umgebung unter Verwendung von Konzepten auf die sich die verschiedenen Entitäten geeinigt haben.

E-Learning-Standards Friesen [Fri05] definiert Standards als dokumentierte Vereinbarungen, die technische Spezifikationen oder andere präzise Kriterien enthalten, die konsistent als Regeln oder Definitionen von Charakteristiken verwendet werden, um sicherzustellen, dass Materialien, Produkte, Prozesse und Services ihren Zweck erfüllen. Im E-Learning-Kontext

werden diese im Entwurf und Implementierung genutzt, um Interoperabilität, Portabilität und Wiederverwendbarkeit von Systemen, Inhalten und Metadaten zu erreichen. Niegemann [Nie08] und Forment et al. [For+09] identifizieren die folgenden Aspekte und Bereiche von E-Learning als Kandidaten für Standardisierung:

- Lernmaterialaustausch: Das Sicherstellen der Portabilität von Inhalts-Paketen sowie deren einheitlicher Verwaltung und Betrieb, hier setzt SCORM [ADL09] an.
- Didaktische Szenarien und Kursabläufe: Die vollständige Spezifizierung der Reihenfolge und Navigation zwischen Sektionen und den beteiligten Rollen und der Tool-Verwendung, ein Standard hierfür ist IMS Learning Design.
- Lerninhalte: Die Spezifikation des internen Aufbaus von Inhalten sowie dessen Semantik, dies ist insbesondere relevant für interaktive Inhalte wie Quizzes. Hier setzen QTI [IMS20] und LTI [IMS19a; Con21] an.
- Metadaten für Suche und Auswahl: Das separate Annotieren von Inhalten mit kategorisierenden Stichworten, der Zielgruppe und Schwierigkeitsgrad, dem Typ des Inhaltes oder dem geschätzten Zeitaufwand kann das Finden ebendieser erleichtern. Voraussetzung ist natürlich ein durchsuchbares Format, ein Beispiel hierfür ist IEEE LOM [IEE02].
- Informationen über Lernende: Die Spezifikation von Techniken um persönliche Daten über die Lernende zu erfassen, dabei kann es sich um logistische Informationen (Namen, Addressen etc.) oder didaktisch relevante Informationen wie Bildungshintergrund oder Präferenzen handeln. Ein Beispiel ist IMS LIP [Con21; IMS03].
- Qualitätsmanagement und Sicherung: Die Spezifikation von Evaluationsmethoden und deren Durchführung.

LTI LTI (Learning Tools Interoperability) [IMS19a; edu13] ist ein Standard für die Integration von E-Learning-Inhalten eines Systems in ein anderes. Es existieren zwei Versionen des Standards: Das hier verwendete, konzeptionell und technisch einfachere LTI 1.1 sowie LTI 1.3 [IMS19b]. Im folgenden wird auf die erste Version eingegangen. Ebenfalls nicht weiter betrachtet werden eine Reihe von optionalen Features des Standards, die implementiert werden können, für die Konformität aber nicht nötig sind - dazu zählen unter anderem der LTI Outcome Service für die Kommunikation der Ergebnisse von Tests und Übungen. Der Ablauf eines LTI Launches, genauer eines *LTI Basic Launches* ist in Abbildung 3.8 dargestellt [IMS19a]. Dieser geht stets vom sogenannten Tool Consumer aus und wird an den Tool Provider gesendet. Bei diesen Systemen handelt es sich in der Regel um Learning Management Systeme, es können allerdings beliebige Systeme sein, solange sie solche Launches verarbeiten können. Das Hauptziel, dass durch diese Launches erreicht werden soll ist die *Identity Assertion* [edu13]. Diese ermöglicht es dem Tool

Parameter	Bemerkungen
lti_message_type	Stets „basic-lti_launch_request“ (nicht optional)
lti_version	Stets „LTI-1p0“ (nicht optional)
Oauth-Parameter	Signatur, Timestamp, Nonce etc.
roles	Die Rollen, die der Nutzer innehat
context_id	Ein transparenter Parameter, der den Launch-Kontext angibt
user_id	Ein transparenter Parameter, der den Nutzer identifiziert
lis_person_name	Der (volle) Name des Nutzers

Tabelle 3.1: Übersicht über die wichtigsten LTI-Parameter [IMS19a; edu13].

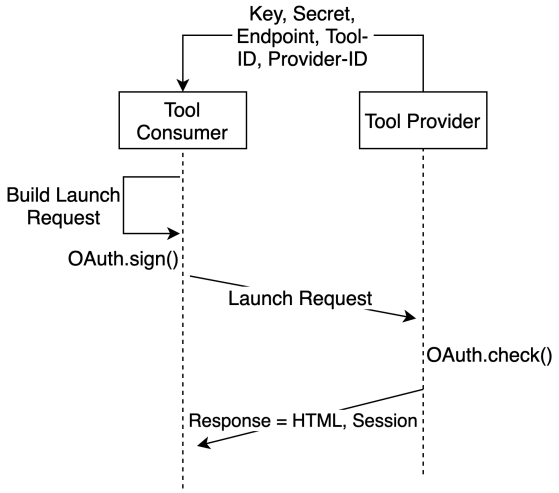


Abbildung 3.8: Der Ablauf eines LTI 1.1 Basic Launch [IMS19a].

Provider einen Nutzer sicher anzumelden und den angefragten Inhalt sichtbar zu machen. Bevor Launches stattfinden können muss die Beziehung zwischen Provider und Consumer etabliert werden. Dies geschieht durch den (von LTI nicht spezifizierten) Austausch eines Consumer Keys und eines Shared Secrets. Mit diesen werden alle ausgetauschten Nachrichten signiert und können so von beiden Parteien jederzeit auf Authentizität und Integrität geprüft werden. Als konkreter Signatur-Algorithmus kommt OAuth 1 zum Einsatz [edu13]. Technisch geschieht der Launch-Request als HTTP POST-Anfrage vom Consumer zum Provider und wird in der Regel als Formular implementiert, das in einer Browser-Umgebung ausgefüllt bzw. gesendet wird. Dies bedeutet wiederum, dass die Antwort des Providers in Form einer Weiterleitung an eine Website geschehen kann und eventuelle Session-Daten weiterverwendet werden können. Es gibt eine Reihe von Parametern, die der Launch Request enthalten kann, einige müssen enthalten sein [IMS19a]. Die wichtigsten Parameter sind in Tabelle 3.1 dargestellt.

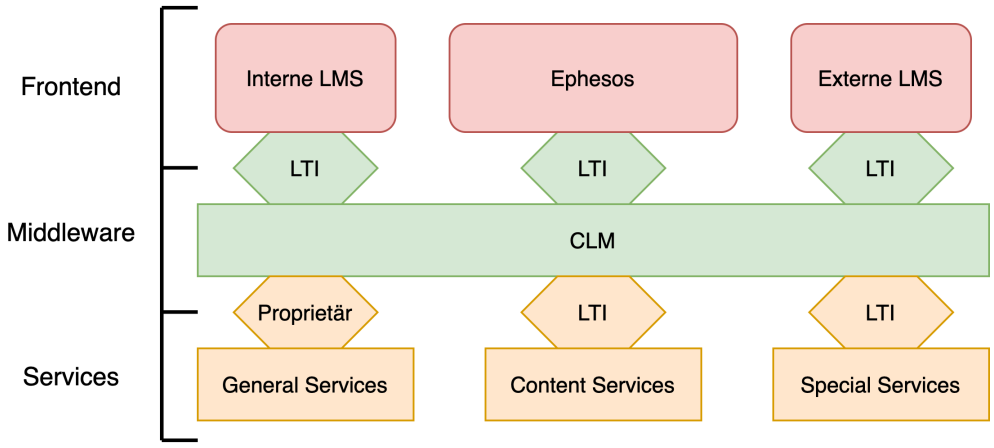


Abbildung 3.9: Die Komponenten der CLM-Architektur [IOS21].

CLM Die CLM (Common Learning Middleware) [IOS21] ist ein am Fraunhofer IOSB entwickeltes Softwaresystem, das es ermöglicht LMS-übergreifende E-Learning-Kurse zu erstellen. Dazu werden in einer Backend-Umgebung verschiedene Lerntechnologien benutzt und gekoppelt. Inhalte, Learning Analytics, Course Enrolments etc. können Plattform-agnostisch bezogen werden, sofern Interoperabilitätstandards wie LTI [IMS19a], QTI [IMS20] oder xAPI [Ini17] verwendet werden. Die Architektur des Systems ist in Abbildung 3.9 dargestellt, in diese soll das zu entwerfende System integriert werden. Sie folgt dem Designparadigma der Serviceorientierung und ist dabei (implizit) in drei Schichten eingeteilt. Die eigentliche Funktionalität ist durch Services implementiert. Diese lassen sich wiederum in drei Kategorien unterteilen [IOS21]:

1. General Services: Von allen Plattformen benötigte Funktionalität, z.B. die User Enrolment Database oder Learning Record Stores für die Persistierung von Lernerfolgen
2. Content Services: LTI-Tools, bei denen es sich nicht um die eigentlichen Learning Management Systeme handelt, also Datenbanken aber auch Authoring bzw. Curation Tools wie das in dieser Arbeit entwickelte sind hier einzuordnen.
3. Special Services: Funktionalität, die ähnlich zu den Content Services ist, aber sich mit einer fokussierteren Domäne beschäftigt, z.B. Recommender-Systeme.

Diese Services sind über eine REST-API erreichbar, die die eigentliche Middleware darstellt. Diese nutzt für die Kommunikation mit LMS oder LRS in den meisten Fällen bestehende Standards wie LTI oder xAPI, sie wird selbst über HTTP angesprochen. Die im Rahmen dieser Arbeit wichtigsten Endpunkte sind konkret: `/uedbs/login` für das Öffnen einer CLM-Session, `/providers/tools` für das Abfragen von den zu durchsuchenden Ankerpunkten

und `/providers/launchdata` für das Initiieren eines LTI Basic Launch. Die Interaktion mit Nutzern findet über das Frontend statt. Hier auch sind die Learning Management Systeme eingebunden. Lernende interagieren mit dem System via des Single-Sign-On-Portals Ephesos. In diesem Portal werden die Kurse bearbeitet, hier sind die Lern-Inhalte und auch Findoo sichtbar bzw. können über LTI geöffnet werden [IOS21].

3.4 Korpusaufbau durch Web Crawling und Deep Crawling

Ein Web-Crawler ist ein Programm, um große Mengen von Websites in kurzer Zeit automatisiert herunterzuladen. Im Rahmen des hier entworfenen Systems wird ein solcher Crawler für das Füllen eines Indexes mit E-Learning-Inhalten verwendet. Diese Sektion fasst die wichtigsten theoretischen Grundlagen dahinter zusammen.

Ziele und Herausforderungen Gemäß Najork und Heydon [NH02] gibt es vier zentrale Anwendungen für Web-Crawling:

1. *Suchmaschinen*: Die Zusammenstellung eines Corpus von Webseiten für den Aufbau eines durchsuchbaren Index.
2. *Web-Archiving*: Das tatsächliche Speichern der Websites.
3. *Web-Analytics*: Die statistische Untersuchung von Webseiten und deren Nutzung.
4. *Web-Monitoring*: Das Sicherstellen von (gesetzlichen) Normen, z.B. das Feststellen von Urheberrechtsverletzungen.

Der Crawler verwaltet eine Menge von URLs, initial ist diese gefüllt mit Seed-URLs. In jeder Iteration wird eine URL ausgewählt, die entsprechende Website wird heruntergeladen und dort eventuell gefundene URLs der Menge hinzugefügt. Der Vorgang ist konzeptionell einfach. Wie Najork und Heydon [NH02] darstelllen, ergeben sich für Real World-Implementierungen allerdings folgende Herausforderungen:

- *Performance und Skalierbarkeit*: Für große Mengen von URLs muss in der Regel auf eine verteilte Architektur zurückgegriffen werden, die das Entfernen und Hinzufügen von neuen Systemen erlaubt. Ebenfalls schwierig ist das Verwalten von URLs und gefundenen Inhalten in Datenstrukturen, die in der Regel größer als der physische Hauptspeicher sind.
- *Politeness*: Die zu durchsuchenden Webserver dürfen nicht überlastet werden, hier ergibt sich eventuell ein Tradeoff mit der Performance des Systems.

- *Kontinuität*: Bereits durchsuchte Websites können sich natürlich ändern, es muss also stets zwischen dem Durchsuchen neuer und bereits bekannter Websites abgewogen werden.
- *Erweiterbarkeit*: Es existiert eine konzeptionelle Trennung zwischen dem Crawler und dessen Aufgabe, d.h. der Verarbeitung der gefundenen Inhalte. Dieser Übergang muss selbst geeignet entworfen werden.

Das Lösen dieser Herausforderungen ist die zentrale Aufgabe der Web-Crawler [NH02]. Dies kann auf vielfältige Art geschehen. Zunächst kann die Menge der verwalteten URLs, das *URL Frontier* durch die Zuweisung von Prioritäten für die URLs und anschließendes Scheduling sowie Duplicate Detection optimiert werden. Für die Kommunikation mit den zu durchsuchenden Servern sind zwei Modelle möglich: Das Push- und das Pull-Modell. Im ersten Fall verteilen die Content-Provider aktiv Inhalte an Aggregatoren, im zweiten müssen sich diese selbst um Inhalte und deren Aktualität kümmern. Das Pull-Modell hat sich durchgesetzt, bedingt durch die Autonomität der Provider, die selbst zu einer geringeren Barrier of Entry führt. Außerdem existieren für das Push-Modell kein Vetrauensmodell und keine standardisierten Protokolle [NH02]. Zu weiteren Kooperationsaspekten zählen Robots-Protokolle, um Inhalte von Crawling auszuschließen und Sitemaps, um das Crawling zu erleichtern. (In dieser Arbeit würden sich LTI Content-Item-Requests [IMS19a] der letzteren Kategorie zuordnen lassen.) Damit einhergehend ist der Umgang mit Gegenspielern, z.B. dem Bereitstellen absichtlich redundanter oder schwer zu durchsuchender Daten [NH02].

Architektur eines Web-Crawlers Die Bestandteile eines generischen Web-Crawlers nach Olson und Najorak [ON10] sind in Abbildung 3.10 dargestellt. Der Reihenfolge nach sind dies:

1. Frontier: Eine Priority Queue, die URLs enthält. Diese bestimmt welche Seite als nächstes untersucht wird, die Priority Queue implementiert also das oben erwähnte Scheduling.
2. HTTP-Fetcher: Dieses Modul übernimmt die DNS-Auflösung, die Prüfung eventueller Restriktionen und den eigentlichen Download.
3. Link Extractor: Hier wird das HTML der Seite geparst und Hyperlinks extrahiert.
4. URL Distributor: Die extrahierten URLs werden an die Crawling-Prozesse verteilt.
5. Custom URL Filter: Eine Blacklist, die das Crawling von bestimmten Seiten ausschliesst.
6. Duplicate URL Eliminator: Bereits besuchte URLs werden hier erkannt und ggf. entfernt.
7. URL Prioritizer: Hier wird entschieden an welcher Stelle in der Prioritätsliste die neue URL eingefügt wird.

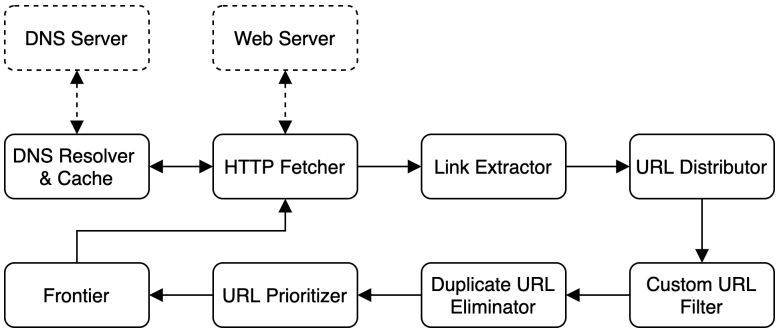


Abbildung 3.10: Die grundlegende Architektur eines Web-Crawlers [ON10].

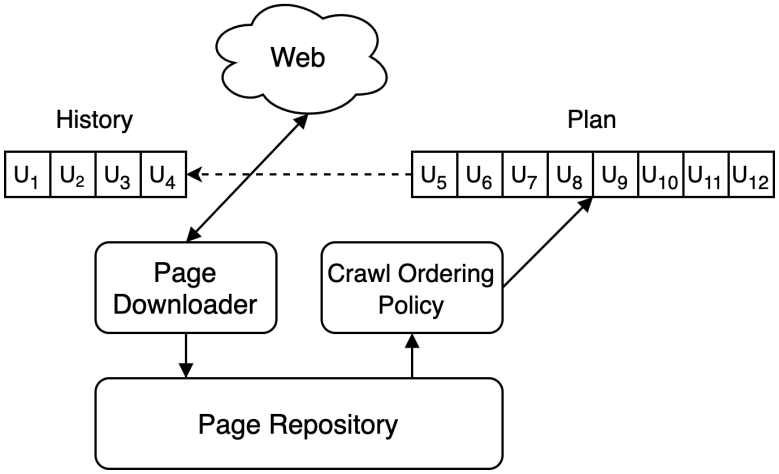


Abbildung 3.11: Das generische Crawling-Modell nach Olston [ON10].

Das Crawl-Ordering-Problem Abbildung 3.11 stellt das grundlegende Crawling-Modell nach Olston [ON10] dar. Die besuchten bzw. noch zu besuchenden URLs werden mit U_i bezeichnet. Im Rahmen dieses Modells existiert ein fundamentaler Tradeoff zischen *Freshness*, der Aktualität der indizierten Inhalte, und *Coverage*, der Breite und Anzahl der indizierten Inhalte. Zur Lösung dieses Tradeoffs gibt es eine Reihe von Crawling-Strategien, die sich grob in inkrementelle Strategien und Batch-Strategien unterteilen lassen [ON10]. Diese werden nachfolgend kurz erläutert.

Batch-Crawling-Strategien Das Ziel jeder Crawling-Strategie ist das Maximieren der *weight coverage WC*. Zum Zeitpunkt t ist diese definiert als:

$$\mathrm{WC}(t) = \sum_{p \in C(t)} w(p)$$

$C(t)$ ist die Menge aller untersuchten Seiten bis zum Zeitpunkt t, jede Seite p hat ein numerisches Gewicht $w(p)$. Als Gewichtsfunktion w wird typischerweise PageRank oder eine Modifikation verwendet [ON10].

Inkrementelle Crawling-Strategien Neben der weighted coverage wird nun analog auch die *weighted freshness* maximiert. Diese ist definiert als:

$$\mathrm{WF}(t) = \sum_{p \in C(t)} w(p) \cdot f(p,t)$$

Die Funktion f gibt die Freshness einer Seite p zum Zeitpunkt t an, der Rest ist definiert wie oben. Ein inkrementeller Crawler hat in jedem Schritt die Wahl entweder eine neue Seite zu untersuchen oder eine bekannte erneut herunterzuladen. Er versucht dabei die durchschnittliche weighted freshness $\overline{WF}$ zu maximieren [ON10]. Es gilt per Definition:

$$\overline{\mathrm{WF}} = \lim_{t \to \infty} \frac{1}{t} \int_0^t WF(t)\, dt.$$

Dieses Optimierungsproblem lässt sich in drei Subprobleme unterteilen [ON10]:

1. Modell-Schätzung: Wie oft wird eine Seite p aktualisiert?
2. Ressourcenallokation: Gegeben eine Crawling-Rate r, wie soll diese auf die einzelnen Seiten p aufgeteilt werden? Die Aufteilung muss dabei stets konsistent sein, d.h. es muss gelten: $\sum_{p \in C} r(p) = r$.
3. Scheduling: Die eigentliche Abbildung der Allokation auf den Crawling-Prozess.

Konkret werden für f Freshness-Modelle untersucht. Diese lassen sich in zwei Kategorien unterteilen [ON10]. Das erste ist das binäre Modell mit $f(p,t) \in \{0,1\}$:

$$f(p,t) = \begin{cases} 1, & p' \text{ im Cache entspricht } p \\ 0, & \text{sonst} \end{cases}.$$

Die zweite Modellklasse lässt sich als kontinuierlich beschreiben. Ein Beispiel ist das Alter einer Seite im Cache:

$$\mathrm{age}(p,t) = \begin{cases} 0, & p' \text{ im Cache entspricht } p \\ a, & \text{sonst} \end{cases}.$$

Der Wert a ist die Anzahl der Prüfungen in denen p und p' sich nicht entsprochen haben, er gibt die Desynchronisierungsrate an. Der eigentliche Vergleich der Version im Cache p' und

der aktuellen Seite p versucht in der Regel nicht tatsächliche Identität festzustellen, da sich Metadaten wie Zeitstempel oder irrelevante Teile wie Werbebanner ändern können, ohne dass p neu geladen werden muss. Mithilfe der Modelle kann man für eine Poisson-Verteilung eine optimale Allokation angeben. Für Details sei wieder auf Olston [ON10] verwiesen. Im Rahmen der aktuellen Arbeit sind diese Überlegungen nur am Rande relevant, der Crawling-Prozess wird hier einfach periodisch im Batch-Modus ausgeführt. Für die Anwendung des Crawlers in großen Systemen können allerdings auch Strategien dieser Art angewendet werden.

Google - der bekannteste Crawler Der bekannteste Crawler im Internet ist für den Index der Google-Suchmaschine verantwortlich, dieser zeichnet sich insbesondere durch seine enorme Skala und Parallelität aus. Eine frühe Form des Crawlers wurde 1998 von Brin und Page [BP98] beschrieben. Konzeptionell besteht dieser aus fünf Komponenten bzw. Prozessen, die als Zyklus aufgefasst werden können:

1. Der URL-Server-Process liest URLs aus Webseiten aus und leitet diese an Crawler-Prozesse weiter.
2. Jeder Crawler-Prozess lädt mehre Dateien via asynchronem I/O parallel herunter, das Crawling läuft also auf zwei Ebenen parallel ab.
3. Die geladenen Dateien werden von StoreServer-Prozessen komprimiert und gespeichert.
4. Ein Indexer liest die gespeicherten Dateien, extrahiert URLs und speichert diese separat.
5. Ein URL-Resolver liest diese URLs, stellt daraus absolute URLs her und speichert diese in URL-Dateien, diese werden wiederum vom Server Prozess gelesen und der Zyklus beginnt erneut.

User-zentriertes Crawling Das Konzept des User-zentrierten Crawlings wird von Pandey und Olston [PO05] vorgestellt. Sie bemerken zunächst, dass Web-Inhalte einer hohen Dynamik unterliegen, der lokale Index (das *Repository*) kann nicht jederzeit für alle Einträge up-to-date sein, es ist also selektives Re-Downloading nötig. Hierbei ist die zentrale Frage die des Scheduling. Dessen Qualität wird von den Autoren durch die User-Quality-of-Service bewertet und hinsichtlich derer optimiert. Die Qualität einer Suchmaschine ist aus Nutzersicht von zwei Faktoren abhängig: Der Eignung der Ranking-Funktion und der *Repository Freshness*, d.h. der Aktualität der (ersten) Ergebnisse. Der zentraler Beitrag der Autoren besteht hierbei in der Schätzung des Einflusses des (Re-)Downloads einer Website auf diese Gütekriterien, ohne diese tatsächlich herunterzuladen und die Nutzung dieser Ergebnisse in einer Scheduling Policy. Implementiert wird dies mithilfe des Query-Logs vergangener Suchen und Messungen der Schwankung in der Qualitäts-Metrik [PO05].

Deep Crawling Hernandez et al. [HRR19] führen eine Survey über aktuelle Entwicklungen im Bereich des Deep Crawlings durch. Sie fangen mit der Begriffsbildung an: Es existieren verschiedene Bedeutungen des Deep Web. Der Begriff bezeichnet aber stets Webseiten, die nicht von Suchmaschinen indiziert werden, z.B. weil sie passwortgeschützt sind, nicht HTML-basiert sind oder schlicht nicht auf sie verwiesen wird und der Crawler sie deshalb initial nicht finden kann. In der Survey wird die Definition von Bergman [Ber01] verwendet: Deep Crawling bezeichnet das Crawling von Websites, die nicht direkt über existierende Links erreichbar sind, sondern als Antwort auf Formular-Eingaben angezeigt oder generiert werden. Entsprechend müssen Deep Crawler im Vergleich zu Standard-Crawlern über zusätzliche Funktionen verfügen [HRR19]. Diese betreffen insbesondere das Entdecken und Ausfüllen von Formularen. Im Allgemeinen ist der Ablauf des Deep-Crawling wie folgt: Zuerst werden für ein gegebene System Entry-Points identifiziert, meistens handelt es sich dabei um Suchmasken. Dies sollte möglichst oft automatisiert stattfinden. Danach findet das *Form-Modeling* statt, das Verstehen von Formular-Komponenten und deren Beziehungen. Nun erfolgt die *Query Selection*, das Auswählen von Suchbegriffen, die konsistent mit den erwarteten Eingaben des Formulars sind. Das anschliessende Ausfüllen der Formulare findet im Rahmen einer Simulation statt, diese ist angelehnt an die Nutzung der Suche durch Menschen. Abschliessend muss eine Form des *Crawling-Path-Learning* stattfinden, die Entwicklung von Strategien, die ein möglichst effizientes und effektives Verfolgen der entstehenden Pfade durch die Deep Web-Inhalte ermöglichen [HRR19].

Im Rahmen dieser Arbeit wird der Begriff des Deep Crawling anders, aber nicht inkompatibel definiert. Diese Arbeit beschäftigt sich mit dem Crawling von Deep Links, d.h. URIs, die direkt auf Inhaltsblöcke in Learning Management Systemen verweisen. Als konkrete Technologie wird LTI [IMS19a] verwendet. Dieser Standard abstrahiert die tatsächliche Problemstellung des Ausfüllens von Formularen. Die E-Learning-Inhalte sind auch Teil des Deep Web wie in der klassischen Definition, sie sind auch nicht Teil von existierenden Suchmaschinen und nur nach Authentifizierung sichtbar. Diese geschieht aber im Rahmen von LTI und der CLM und nicht über das tatsächliche Ausfüllen von Nutzer/Passwort-Formularen. Für die oben erwähnten Schritte des Deep Crawling werden ebenfalls analoge Komponenten entworfen, beispielsweise das Crawling Path Learning durch die Kombination eines URL-Frontier mit Black- und Whitelists.

3.5 Information Retrieval und Indexstrukturen

Neben dem Crawler ist spielt das Information Retrieval auf einem Index eine entscheidende Rolle im Backend des entworfenen Systems. Manning et al. [MRS08] definieren Information

Retrieval als das Finden von unstrukturierten Materialien, die ein gegebenes Informationsbedürfnis befriedigen, in der Regel in einer großen Menge von Inhalten. Der Entwurf von Information Retrieval Systemen und deren Evaluation findet in dieser Arbeit in diesem theoretischen Rahmen statt. Als Antwort auf ein *Informationsbedürfnis* $q \in Q$ liefert das System eine Menge von Dokumenten $R_q \subseteq R$, die *Ergebnisse*. Das Konzept des Informationsbedürfnis beschreibt den gesuchten Inhalt, dieser wird von Nutzerseite in Form von Anfragen an das System übergeben. Bei diesen Anfragen kann es sich um kontextfreie Zeichenketten oder (insb. bei Recommender-Systemen) um komplexere Konstrukte mit Kontext handeln. Die Unterscheidung zwischen dem gesuchten Inhalt und den eingegebenen Anfragen bildet den Umstand ab, dass der Nutzer in der Regel nicht die informationstheoretisch optimalen Suchbegriffe verwenden wird, das System aber auch damit umgehen können muss. Im Rahmen dieses Frameworks wird davon ausgegangen, dass es sich bei den Dokumenten und den Suchbegriffen um Text handelt [MRS08].

Die Realisierung von Information Retrieval-Algorithmen kann auf viele Arten geschehen [MRS08]. Ein erster naiver Ansatz ist es die Menge aller in den Dokumenten vorkommenden Terme in einem Vokabular zu formulieren und in einem Matrix-Format zu speichern, ob ein gegebenes Dokument für einen Suchbegriff relevant ist. Der erhebliche Platzverbrauch und die fehlende Flexibilität verhindern jedoch eine richtige Implementierung. Stattdessen verfolgen fast alle Systeme den Ansatz eines inversen Index: Eine Datenstruktur, die für jedes Wort des Vokabulars die Dokumente speichert in denen es vorkommt, in der Regel durch Pointer. Der Index wird beim Einfügen neuer Dokumente aktualisiert bzw. für die Suchphase vorberechnet. Dieser Index ist sehr viel kleiner als die Menge der gespeicherten Dokumente und kann in der Regel im Speicher gehalten werden. Nun kann bereits boolesches Information Retrieval durchgeführt werden: Durch einen Lookup im Index für ein gegebenes Wort kann die Menge der relevanten Dokumente bestimmt werden. Ist ein Wort nicht im Index, wird die leere Menge zurückgegeben. In Real-World-Szenarien kann und muss aber noch eine Reihe von Optimierungen und Erweiterungen stattfinden [MRS08]. Dazu zählen:

- Eine Erhöhung der Effizienz durch Caching-Strategien sowie schnellere Lookups und Dokumentenzugriffe .
- Die Implementierung von flexibleren Lookups durch eine NLP-Analyse von Dokumenten und Anfragen, hier muss mit Singular und Plural, Präfixen und Suffixen, Konjugationen, Zeitformen etc. umgegangen werden können.
- Ranking: In der Regel sind nicht alle Dokumente gleich relevant - kommt ein Term mehrfach vor, ist das Dokument relevanter als wenn er nur einmalig vorkommt, kommt ein Term insgesamt selten in der Menge der Dokumente vor, sind dessen Vorkommnisse

wichtiger als die eines häufigen Terms etc. Für die Erstellung solcher Rankings muss eine Menge von Metadaten zusätzlich zu den Pointern im Index verwaltet werden.
- Andere Anfrage-Arten: Oft besteht die Anfrage nicht aus einem einfachen Keyword, sondern aus einer Menge von Keywords, logischen Verknüpfungen und Constraints, oder es handelt sich gar nicht um Text-Daten wie z.B. beim vektorbasierten Information Retrieval. Für alle diese Fälle muss der Index erweitert oder verändert werden.

In Kapitel 6 werden die Komponenten des entwickelten Systems anhand von Hypothesen evaluiert. Der Erfüllungsgrad der Hypothesen wird anhand von bekannten Information Retrieval-Metriken geprüft, die hier nachfolgend motiviert und erläutert werden.

Präzision Die von Information Retrieval Systemen zurückgegebenen Ergebnisse müssen korrekt sein. Für Suchmaschinen bedeutet das konkret, dass als Antwort auf eine Anfrage viele relevante Dokumente und wenig irrelevante Dokumente angezeigt werden. Ein Maß für die Korrektheit ist die *Präzision*, definiert in Gleichung 3.1 in der Formulierung von Manning et al. [MRS08]. Sei $R_q^\star$ die Menge der relevanten Dokumente und R_q die Menge aller gelieferten Dokumente, dann ist die Präzision das Verhältnis der *true positives* (relevante Dokumente, die geliefert wurden) zu allen gelieferten Dokumenten, *true positives* und *false positives*. Der Wertebereich ist folglich [0,1].

$$\text{precision}(q) := \frac{|R_q^\star|}{|R_q|} = \frac{tp}{tp + fp} = P(r \in R_q^\star \mid r \in R_q) \tag{3.1}$$

In dieser Formulierung ist die Präzision eine mengentheoretische Metrik. Moderne Suchmaschinen liefern in der Regel aber nicht eine ungeordnete Menge, sondern (auch) ein Ranking der Ergebnisse. Um damit umzugehen kann das Konzept der Präzision erweitert werden. Ein naheliegender Ansatz ist dabei *precision@k*, also die Präzision der ersten k gelieferten Ergebnisse oder das Konzept der *Precision-Recall-Curve*, die weiter unten erläutert wird [MRS08].

Recall Neben der Korrektheit der gelieferten Ergebnisse ist deren Vollständigkeit wichtig - wenn sich relevante Inhalte im Index befinden, ist es die Aufgabe des Systems diese auffindbar zu machen. Die klassische Metrik dazu ist der *Recall*, wieder in der Formulierung von Manning [MRS08], gegeben via Gleichung 3.2.

$$\text{recall}(q) := \frac{|R_q|}{|R_q^\star|} = \frac{tp}{tp + fn} = P(r \in R_q \mid r \in R_q^\star) \tag{3.2}$$

Der Recall ist also der Anteil der relevanten Dokumente, die auch geliefert wurden im

Vergleich zu allen relevanten Dokumenten (bestehend aus den *true positives* und den *false negatives*). Analog zur Präzision ist der Wertebereich [0,1] und es lässt sich hier ebenfalls der *recall@k* definieren.

Precision-Recall-Kurven Die Betrachtung der ersten k Dokumente für Qualitätsmetriken ist instabil und stark vom unterliegenden Korpus abhängig [MRS08]. Deswegen wird zur Evaluation oft auf ein explizites k verzichtet und stattdessen Präzision und Recall in Verbindung gesetzt. Dies geschieht mit den Precision-Recall-Kurven. Hierbei wird für eine Menge an Anfragen Q an ausgewählten Recall-Grenzwerten die Präzision gemessen und das Ergebnis als Funktion dargestellt. Oft werden die 11 Grenzwerte $0{,}0.1 \ldots 1$ gewählt, dargestellt in Gleichung 3.3 [MRS08].

$$P = \frac{1}{11} \sum_{i=0}^{i=10} \frac{1}{|Q|} \sum_{j}^{|Q|} precision_j(r_i) \tag{3.3}$$

$precision_j(r_i)$ bezeichnet die Präzision für Suche j an den Recall-Punkten $r_i = j/10, r_0 = 0$, $\ldots, r_{10} = 1$. Da der Graph nur an den Stützpunkten definiert ist, muss dazwischen interpoliert werden. Dies geschieht mit der folgenden Vorschrift [MRS08]:

$$precision_j(r_i) = \max_{r_l \geq r_i} precision_j(r_l)$$

ROC-Kurven Neben Precision-Recall-Kurven sind ROC-Kurven eine etablierte Darstellung der Performance eines Information Retrieval-Systems. ROC steht hierbei für *receiver operating characteristics*. Diese stellen die Sensitivität $\text{sensitivity} = tp/(tp + fn)$ auf der y-Achese in Abhängigkeit zur False-Positive-Rate auf der x-Achse dar. Diese ist gleich zu $1 - specificity$, wobei die Spezifizität als $\text{specificity} = fp/(fp + tn)$ definiert ist [MRS08]. Die Fläche unter dieser Kurve, genannt ROC-AUC (area under curve) ist ebenfalls eine Metrik für die Qualität des Systems mit dem Wertebereich [0,1]. Die ROC-Kurve eines guten Systems steigt (insbesondere am Anfang für eine hohe Spezifizität) schnell an [MRS08].

Mean Average Precision Für eine Mengen an Anfragen Q ist die durchschnittliche Präzision ein naheliegendes und weitverbreitetes Qualitätsmaß, sie ist gegeben in Gleichung 3.4 [MRS08]. Wenn für eine Anfrage $q_j \in Q$ die Dokumente $\{d_1, \ldots d_{m_j}\}$ relevant sind, ist R_{jk} dabei die Menge der geordneten, zurückgegebenen Dokumente bis d_k erreicht wird. Wird ein Dokument gar nicht zurückgegeben, so ist die Präzision precision als 0 definiert.

$$\text{map}(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{precision}(R_{jk}) \tag{3.4}$$

Die Mean Average Precision kann als eine (grobe) Approximation der Fläche unter der Precision-Recall-Kurve gesehen werden, siehe Manning et al. [MRS08].

F-Score Um die Metriken der Präzision und des Recalls in einer zu kombinieren, wird oft der F-Score verwendet. Dieser ist definiert als der gewichtete, harmonische Mittelwert der beiden Metriken. Er ist gegeben in Gleichung 3.5 mit den Parametern $\alpha \in [0,1]$ und $\beta^2 = \frac{1-\alpha}{\alpha}$, somit gilt $\beta \in [0,\infty)$ [MRS08].

$$F(q) \coloneqq \frac{1}{\frac{\alpha}{\text{precision}(q)} + \frac{(1-\alpha)}{\text{recall}(q)}} = \frac{(\beta^2+1)\cdot \text{precision}(q)\cdot \text{recall}(q)}{\beta^2 \cdot \text{precision}(q) + \text{recall}(q)} \tag{3.5}$$

Für $\beta = 1$ (und $\alpha = 1/2$) ergibt sich der balancierte F_1-Score. Dieser vereinfacht die obere Formel zu Gleichung 3.6.

$$F_{\beta=1}(q) \coloneqq \frac{2\cdot \text{precision}(q)\cdot \text{recall}(q)}{\text{precision}(q) + \text{recall}(q)} \tag{3.6}$$

Als Mittelwert von Metriken mit Wertebereich $[0,1]$ hat der F-Score denselben Wertebereich.

Normalized Discounted Cumulative Gain Wenn die Relevanz eines Dokumentes nicht binär gegeben ist, können die bisherigen Metriken nicht ohne Weiteres angewendet werden. Die Metrik des Normalized Discounted Cumulative Gains ist aber insbesondere auch hier anwendbar [MRS08]. Sei $R(j,d)$ die Relevanz des Dokuments d für Suche j, die Metrik ist dann gegeben in Gleichung 3.7.

$$\text{ndcg}(Q,k) = \frac{1}{|Q|}\sum_{j=1}^{|Q|} Z_k \sum_{m=1}^{k} \frac{2^{R(j,m)}-1}{\log(1+m)} \tag{3.7}$$

Z_k ist ein Normalisierungsfaktor, sodass ein perfektes Ranking einen NDCG von 1 hat.

3.6 Webtechnologien

Diese Sektion gibt eine Übersicht über die Technologien und Umgebungen, die für die Implementierung des Systems verwendet werden.

Node.js-Umgebung Der Industriestandard für das Backend von Web-Applikationen ist Node.JS.[1] Dieser besteht zum einen aus einer Laufzeitumgebung für Javascript, die auf der

1 https://nodejs.org/en/

Google Chrome V8 Engine basiert und es ermöglicht JavaScript auch ohne Browser auszuführen. Zum anderen besteht Node aus NPM, dem *Node Package Manager*. Dieser ermöglicht die Installation, Aktualisierung und Verwaltung von Modulen, die von Dritten entwickelt wurden und nun für andere Projekte genutzt werden können, was den Entwicklungsaufwand erheblich reduziert. Für diese Arbeit wird Node.JS als Server-Umgebung, für die LTI-Funktionalität, Teile des Recommender-Moduls und im Rahmen des Crawlers genutzt.

Angular Für das Frontend und die Interaktion des Nutzer mit dem Assistenzsystem wird Angular[2] verwendet. Dabei handelt es sich um ein Framework für die Entwicklung von Web-Applikationen. Die Verwendung eines Frameworks stellt die Performance der fertigen Applikation sicher, reduziert den Entwicklungsaufwand durch vorgefertigte Implementationen von häufig gebrauchter Funktionalität und das Automatisieren von Build-Prozessen, Internationalisierungen etc. Angular basiert auf TypeScript, eine von Microsoft entwickelte Erweiterung von JavaScript, die ein Typsystem integriert, dabei aber vollständig rückwärtskompatibel ist und zu JavaScript kompiliert wird. Im Rahmen von Angular gibt es eine Reihe von zentralen Konzepten wie die Kopplung von HTML und der zugehörigen Logik in Komponenten sowie der Verwendung von Services für komponentenübergreifende Logik. Desweiteren gibt es eine Reihe von Best Practices wie die ubiquitäre Anwendung des Observer-Patterns und eine Integration von RxJS in die Applikation. Im Rahmen der Arbeit wird Angular konkret für den Editor sowie das Interface für die Suchmaschine und das Recommender-System verwendet.

Web-Crawling in einem Headless Browser Es gibt eine Reihe von Bibliotheken für das Crawling von Webseiten, berühmte Beispiele aus dem Python-Ökosystem sind *Scrapy* und *Beautiful Soup*. Diese Werkzeuge funktionieren durch das Anfragen von Web-Seiten durch HTTP-Get-Requests und das anschließende Parsen der HTML-Struktur. Für komplexere Web-Seiten, die Features wie Ajax-Calls, endloses Scrolling etc. verwenden, sind diese Werkzeuge allerdings nicht geeignet. Für ein robustes Crawling muss also ein Browser verwendet werden, da der Aufbau der in dieser Arbeit zu durchsuchenden Systeme unbekannt ist und folglich die genannten Limitierungen eine Rolle spielen könnten. Hier kommt *Puppeteer*[3] zum Einsatz. Dabei handelt es sich um eine JavaScript-Bibliothek, die es ermöglicht eine Chromium-Instanz zu kontrollieren, also eine Open-Source-Version des Google Chrome Browsers. Diese kann auch headless verwendet werden, d.h. ohne die grafische Darstellung des Interfaces und der Seite.

2 https://angular.io
3 https://developers.google.com/web/tools/puppeteer

Elasticsearch Elasticsearch [DG13] ist eine Suchmaschine, die in einer Server-Umgebung läuft und via HTTP ansprechbar ist. Sie erlaubt das Verwalten von (in-memory) Indizes, also Sammlungen von Dokumenten eines Schemas. Unter einem Schema (oder *Mapping*) versteht man in diesem Kontext eine Definition von den in den Dokumenten vorhandenen Feldern und deren Typen. Elasticsearch ist auch nativ dafür geeignet in einer verteilten Umgebung auf mehreren Servern betrieben zu werden und bietet zu diesem Zweck Techniken wie das Sharding von Indizes. Die Suchmaschine basiert auf der Apache Lucene Engine, als solche ist sie sehr performant, bietet eine state-of-the-art Implementierung des TFIDF-Suchalgorithmus [Ram+03] und unterstützt Query Strings, d.h. Standard-Suchkonzepte wie logische Verknüpfungen, exakte Matches etc [DG13]. Für die Zwecke dieser Arbeit relevant ist außerdem das Feature des Inline-Scripting, das es erlaubt für eine Suchanfrage komplexe Ranking-Scripte in der Sprache *painless* zu verwenden und eine native Unterstützung für vektorbasierte Anfragen via der Cosinus-Ähnlichkeit.

Container Unter einem Container versteht man eine Applikation, die in einer isolierten, portablen Umgebung lauffähig ist. Ein Container enthält den Code der Applikation, alle ihre Abhängigkeiten sowie Einstellungen für das Deployment wie Startup-Skripte oder welche Ports freizugeben sind. Dies wird grundlegend durch einen auf dem eigentlichen Betriebssystem laufenden Host-Service ermöglicht, der ein virtuelles Datei-System verwaltet. Vorteile dieses Ansatzes sind das zuverlässige Ausführen von Applikationen ohne Installation und mit minimaler Konfiguration sowie die Reproduktion von Umgebungen. In dieser Arbeit wird das System als eine Menge von Containern entworfen und ausgeliefert und nutzt selbst Container von Applikationen wie dem NginX-Server. Als konkrete Technologie für die Container wird Docker[4] verwendet.

3.7 Natural Language Processing

Wie Nadkarni et al. [NOC11] bemerken, überschneiden sich die Felder des Information Retrieval und Natural Language Processing (NLP) zunehmend. Jegliche semantische Suchfunktionalität muss via einer Form von NLP implementiert werden, diese Sektion gibt einen kurzen Überblick über die hier relevanten Techniken.

Natural Language Processing und Neuronale Netze Für Probleme in der Disziplin des Natural Language Processing ist die Eingabe eine Reihe von Worten in einer natürlichen

4 https://www.docker.com

Sprache, die mit diesen zu lösenden Aufgaben sind sehr vielseitig. Auf einer niedrigen Abstraktionsebene sind diese in der Regel: Sentence Boundary Detection, Tokenization, Part-of-speech tagging und morphologische Dekomposition [NOC11]. Auf einer mittleren Abstraktionsebene sind die Aufgaben zum Beispiel Named Entity Recognition oder Relationship Extraction. Auf der höchsten Ebene ist die eigentliche Anwendung zu finden - das Übersetzen von Text, die Generierung von Text oder die hier relevanten semantischen Einbettungen [NOC11]. Unabhängig vom Anwendungsfall müssen Abhängigkeiten zwischen Tokens bzw. Entitäten modelliert werden. Dafür ist immer eine Art von Speicher und der Einfluss von früheren Tokens in der Sequenz auf spätere notwendig. Neuronale Netze und Deep Learning sind die aktuell wichtigste Technik des maschinellen Lernens und sind auch hier sehr vielseitig einsetzbar. Zu den wichtigsten Netzwerk-Architekturen, die die NLP-Anforderungen erfüllen zählen Recurrent Neural Networks, LSTM-Networks und Transfomer-Netzwerke. Letztere bieten die beste Performance und werden nachfolgend genauer erläutert.

Die Transformer-Architektur wurde erstmals von Vaswani et al. [Vas+17] formuliert, eine grobe Übersicht ist in Abbildung 3.12 gegeben. Sie ist besteht aus einer Encoder-Decoder-Struktur, wobei Encoder und Decoder aus sechs identischen Ebenen bestehen. Der Encoder bildet eine Input-Sequenz $x_1, \ldots, x_n$ auf eine stetige Repräsentation dieser ab, notiert als $z_1, \ldots, z_n$. Der Decoder bildet diese wiederum auf eine Output-Sequenz $y_1, \ldots y_n$ ab. Hierbei werden für die Berechnung von y_i sowohl $z_0, \ldots, z_i$ als auch $y_1, \ldots, y_{i-1}$ verwendet [Rus18]. Der zentrale Fortschritt der Transformer-Architektur ist das Konzept der Multi-Head Self Attention. Die Details dieser Funktion werden hier nicht weiter diskutiert, sie ermöglicht es aber eine der zentralen Herausforderungen in der Modellierung von natürlicher Sprache zu lösen: *long-term dependencies*. Werden Sätze streng der Reihenfolge nach verarbeitet, muss Kontext ggf. über sehr viele Tokens persistiert werden, um die Semantik eines Satzes zu erfassen. Außerdem erschwert diese Art der sequenziellen, Wort-basierten Modellierung die Parallelisierung. Mit Multi-Head Attention können die Repräsentationen vieler Tokens unabhängig ihrer Position gewichtet betrachtet werden [Vas+17]. Die Transformers-Bibliothek [Wol+20] erlaubt die Verwendung von bereits konstruierten und trainierten Modellen mit dieser Architektur.

BERT BERT (Bidirectional Encoder Representations from Transformers) ist ein von Devlin et al. [Dev+19] entwickeltes Deep Learning Modell, das auf der oben erwähnten Transformer-Architektur basiert. Es wurde entworfen, um Modelle nach dem Pre-Training möglichst einfach, dass heißt ohne Änderung der Architektur, an einen gegebenen Anwendungsfall anpassbar zu machen. Die Architektur ist in Abbildung 3.13 dargestellt. Es handelt sich um einen bidirektionalen Transformer, was bedeutet, dass die Technik der *bidirektionalen Self-Attention* angewendet wird und somit für ein gegebenes Token sowohl der linke als auch der rechte Kontext be-

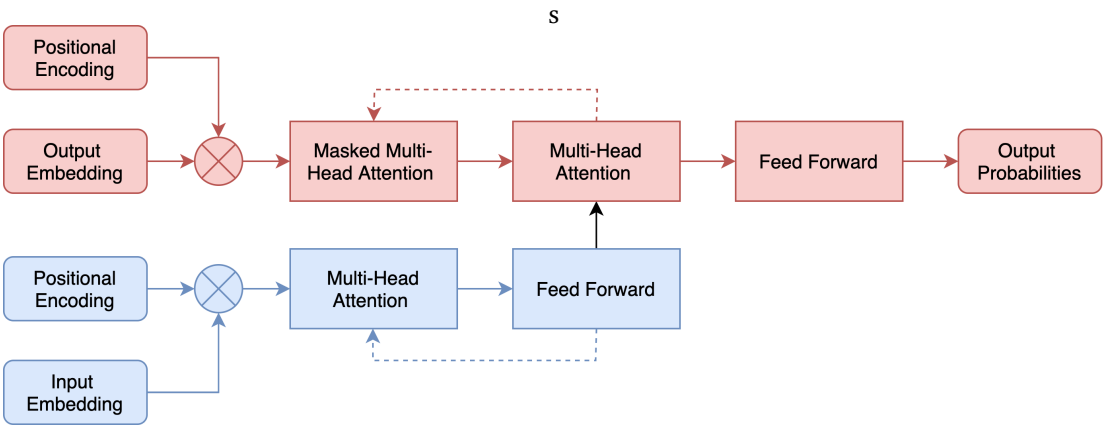


Abbildung 3.12: Übersicht über die Transformer-Architektur, modifiziert nach [Vas+17].

trachtet werden kann. Das Modell kann Eingaben flexibel repräsentieren und kann einen oder zwei Sätze beliebiger Länge entgegennehmen [Dev+19]. Satz bedeutet hier nicht (nur) einen linguistischen Satz, sondern eine beliebige Folge von Text. Jeder Satz enthält ein spezielles CLS-Token für Klassifikationsaufgaben. Die beiden Sätze sind durch ein SEP-Token getrennt, außerdem enthält die Repräsentation jedes Tokens die Information über den Ursprungssatz sowie die Einbettungen des Tokens, der Position und des Segments. Das Pre-Training des Modells erfolgt auf dem BookCorpus mit 800 Millionen Worten und dem englischen Wikipedia mit 2.5 Milliarden Worten. Es besteht aus zwei Aufgaben [Dev+19]:

- *Masked LM*: Für jeden Satz im Trainingsset wird ein bestimmter Anteil der Tokens verdeckt und das Modell muss diese vorhersagen. Das Masking ist nötig, da das Modell bidirektional funktioniert und sich bei konventionellem Next-Token-Training in einem Multi-Layered-Kontext selbst sehen und die Tokens somit trivial ablesen könnte.
- *Next Sentence Prediction*: Um die Beziehungen zwischen Sätzen zu lernen, werden aus dem Trainingsset Satz-Paare generiert. Mit einer Wahrscheinlichkeit von 50% sind die Sätze Nachbarn, mit 50% nicht und das Modell muss dies korrekt klassifizieren.

Das Fine-Tuning des Modells funktioniert durch simples End-to-End-Training und ist sehr viel kostengünstiger als das Pre-Training [Dev+19]. Den Trainingsdaten müssen hierbei aber natürlich die korrekten entsprechenden Tokens zugewiesen werden, abhängig von der Art der Aufgabe.

Sentence-BERT Reimers und Gurevych [RG19] erweitern und adaptieren BERT. Sie merken zunächst an, dass das klassische BERT-Modell zwar sehr gute Performance für Klassifikationen und Satz-Fortführungen bietet, nicht aber für Anwendungen im Rahmen von semantischen

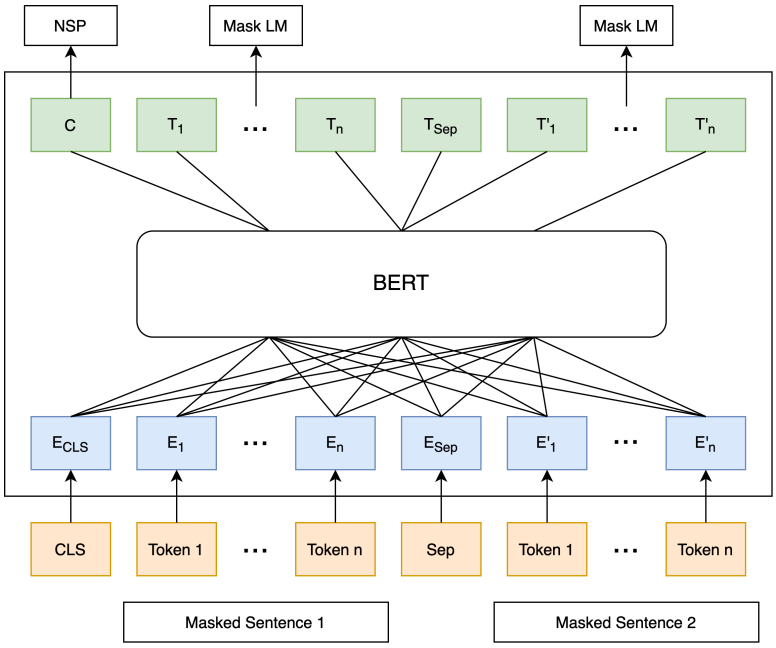


Abbildung 3.13: Die Architektur von BERT [Dev+19].

Suchen und Clustering. Die naheliegende Adaption für diese Anwendungen besteht darin den Output (also die Token-Repräsentationen) des BERT-Modells auf feste, satzweise Einbettungen in einem Vektor-Raum abzubilden, entweder aus dem CLS-Token oder durch die Aggregation der Gewichte des vorletzten Layers [RG19]. Diese Methode führt allerdings zu Ergebnissen, die schlechter als viel primitivere Ansätze wie GloVe [PSM14] sind. Deshalb wurde eine neue Architektur, die siamesische Netz-Architektur angewendet, dargestellt in Abbildung 3.14. Diese ermöglicht es nativ Vektor-Einbettungen für ganze Sätze zu berechnen, die dann via Metriken wie der Cosinus-Distanz semantisch verglichen werden können [RG19]. Erreicht wird dies durch das Hinzufügen einer Pooling-Operation zu dem Standard-BERT-Modell und das Verknüpfen der Gewichte der Modelle in der siamesischen Architektur, hierbei wird eine Loss-Funktion angewendet, die die semantische Vergleichbarkeit der Einbettungen sicherstellt, z.B. die MSE-Funktion für die Cosinus-Distanz der Einbettungen [RG19].

Ontologien und DBPedia Gauch [GCP03] definiert eine Ontologie als eine formelle, explizite Spezifikation einer gemeinsamen Konzeptualisierung. Sie gibt einer gegebenen Domäne also ein gemeinsames Vokabular und eine Beschreibung der Bedeutung und Beziehungen der enthaltenen Terme. In den für diese Arbeit relevanten Feldern des Web-Crawling und Information Retrieval können Ontologien als zusätzliches Werkzeug benutzt werden, um

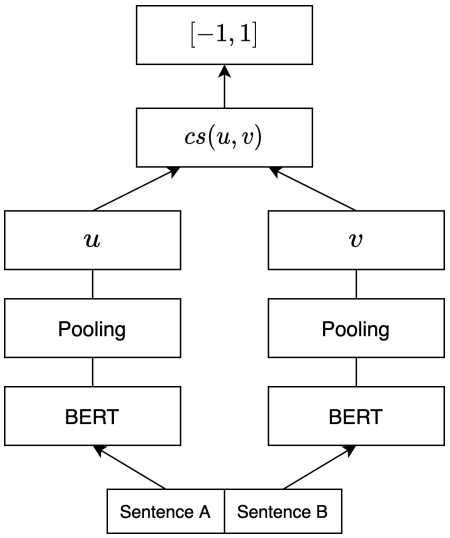


Abbildung 3.14: Der Inferenz-Ablauf von Sentence-BERT [RG19].

ein semantisches Verständnis für Inhalte zu erlangen. Die Anwendung von Ontologien, zum Beispiel als Teil des *Semantic Web* wurde bereits in Kapitel 2 erwähnt. Relevant für das zu entwerfende System sind General-Purpose-Ontologien: Solche, die ein Vokabular für möglichst viele inhaltliche Domänen enthalten. Entsprechend sind diese aber meist *shallow*, enthalten also für ein gegebenes Thema weniger Einträge und die Beziehung zwischen Domänen sind nur grob modelliert, z.B. durch flache Hierarchien. Dies macht ein Reasoning, also das (logische) Ableiten von Beziehungen zwischen Konzepten oder Entitäten mit diesen Ontologien schwer, diese Thematik wird in den Kapiteln 4, 5 und 6 noch genauer erläutert. Bei der in dieser Arbeit verwendeten General-Purpose-Ontologie handelt es sich um DBPedia [Leh+15], eine aus dem Text-Corpus von Wikipedia mittels Linked Data-Technologien abgeleitete Ontologie. Jeder Eintrag in der Ontologie enthält eine Menge von Attributen wie den Typ der zum Eintrag gehörenden Entität, den Abstract des zugehörigen Artikels, ein- und ausgehende Referenzen etc. Für das hier entworfene System sind insbesondere die assoziierten Konzepte relevant mit denen jede Entität annotiert ist. Die Inhalte sind mittels RDF und SPARQL-Endpoints abfragbar [Leh+15].

4 Konzeptionierung des Assistenzsystems

Dieses Kapitel stellt den konzeptionellen Entwurf des Assistenzsystems dar. Dieser steht stets vor dem Hintergrund der Interoperabilität und Robustheit. Das System muss mit mehreren anderen Systemen kommunizieren, sich dabei an Standards (und deren individuellen Ausprägungen) halten sowie stabil gegenüber Änderungen sein. Falls sich größere Änderungen ergeben, muss das System darüber hinaus einfach modifizierbar bzw. erweiterbar sein. Aus konzeptioneller Sicht besteht das System aus zwei Komponenten: Dem Backend, zusammengesetzt aus Deep Crawler und Index, dem Suchalgorithmus und des Recommender-Systems sowie dem Frontend, das die Nutzer-Interaktion übernimmt und zu diesem Zweck algorithmisch mit Kurs-Repräsentationen umgehen muss und das (korrekte) Delegieren von Funktionalität an das Backend übernimmt. In Sektion 4.1 findet eine detailliertere Anforderungsanalyse statt. Sektion 4.2 gibt eine Übersicht über den Web-Crawling-Algorithmus und deren Eigenschaften. In Sektion 4.3 wird beschrieben wie sich die erhobenen Anforderungen auf das Design des Index auswirken. Sektion 4.4 stellt den Entwurf des Suchalgorithmus dar. Sektion 4.5 beschreibt schließlich die Integration eines Recommender-Systems.

4.1 Anforderungsanalyse und Szenarien

Abbildung 4.1 gibt einen Überblick über die Subsysteme bzw. Aufgaben des zu entwerfenden Assistenzsystems und die Rollen der mit ihnen interagierenden Nutzer. Ziel des Systems ist das Indizieren von E-Learning-Inhalten, die in via der Common Learning Middleware (CLM) [IOS21] verbundenen LMS vorliegen, um diese via einer Suche auffindbar zu machen. Diese Suche wiederum kann von Lernenden unmittelbar verwendet werden oder von Lehrenden, um die Inhalte in LMS-unabhängigen Kursen einzubinden. Um möglichst viele Inhalte zu finden, soll eine semantische Suche implementiert und ein Recommendersystem in den Editor integriert werden. Als dritte Rolle lassen sich Administratoren identifizieren, die die LMS aufsetzen und deren Integration in die CLM und die anschließende Konfiguration übernehmen.

Das Assistenzsystem besteht aus konzeptioneller Sicht aus zwei Komponenten. Die erste ist das User Interface, dieses ist für die Interaktion mit Lehrenden und Lernenden verantwortlich. Hier können Suchanfragen gestellt, Kurse verarbeitet und Optionen gesetzt werden, das

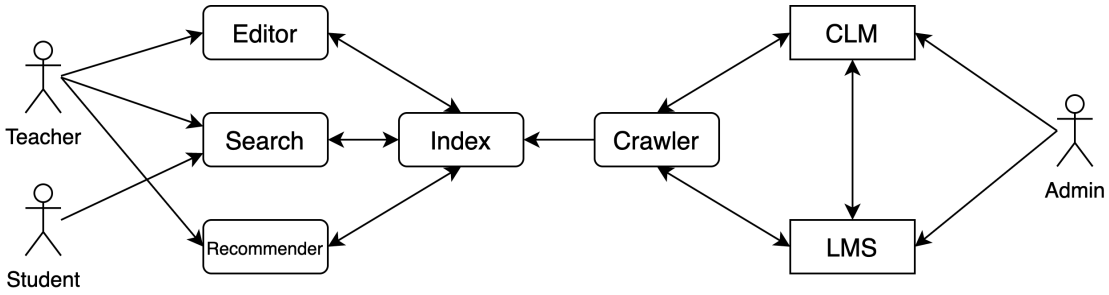


Abbildung 4.1: Die Rollen und Aufgaben des zu entwerfenden Systems.

Nutzungskonzept ist in Abbildung 4.2 dargestellt. Die konkrete Implementierung dieser Funktionalität findet in der Server-Komponente des Systems statt. Hier wird ein Index verwaltet, der die Such- und Recommenderfunktionen ermöglicht. Dieser wird wiederum mit Inhalten aus Learning Manegement Systemen gefüllt, die von einem Deep Crawler in diesen gefunden werden. Die zentrale Anforderungen, die das System erfüllen muss, sind somit die folgenden:

- Die Kommunikation mit der CLM, um Zugriff auf die LMS zu erhalten und ein robustes Crawling derselben
- Ein Indizieren der gefundenen Inhalte, um eine Volltextsuche und eine semantische Suche anhand von Keywords zu erlauben
- Eine Präsentation der Ergebnisse an den Nutzer im Rahmen eines Editors durch die Abbildung der Suchergebnisse auf den IMS Common Cartridge-Standard
- Die Integration eines Recommender-Systems in diesen Editor für proaktive Vorschläge

Der Fakt, dass es sich bei den indizierten Inhalten um E-Learning-Materialien handelt ist relevant für das Crawling, für die Information-Retrieval-Komponenten hat es aber nur geringen Einfluss. Die zentrale Erkenntnis ist hierbei, dass (nahezu) alle Inhalte des Internets auch E-Learning-Inhalte sind bzw. sein können. Insbesondere ist zu beachten, dass der semantische Inhalt, die *Domäne*, beliebig ist. Auch inhaltsunabhängige Charakteristiken sind schwer zu identifizieren. Es lassen sich nur ein paar (schwache, allgemeine) Eigenschaften festhalten:

- Inhaltlicher Bezug: nicht festgelegt
- Textkategorien: In der Regel Sachtexte (*non-fiction*)
- Textursprung: mehr Fachartikel als News/Webdaten
- Textlänge: kurze Texte in Aufgabenstellungen oder Fragen, durchschnittlich lange Texte in interaktiven Modulen o.ä. oder sehr lange Texte in (externen) Dokumenten

Beim System handelt es sich also um ein *General-Purpose* Information Retrieval System. Das entworfene System legt den Fokus auf natürliche Sprache. Mathematische Formeln, Videos

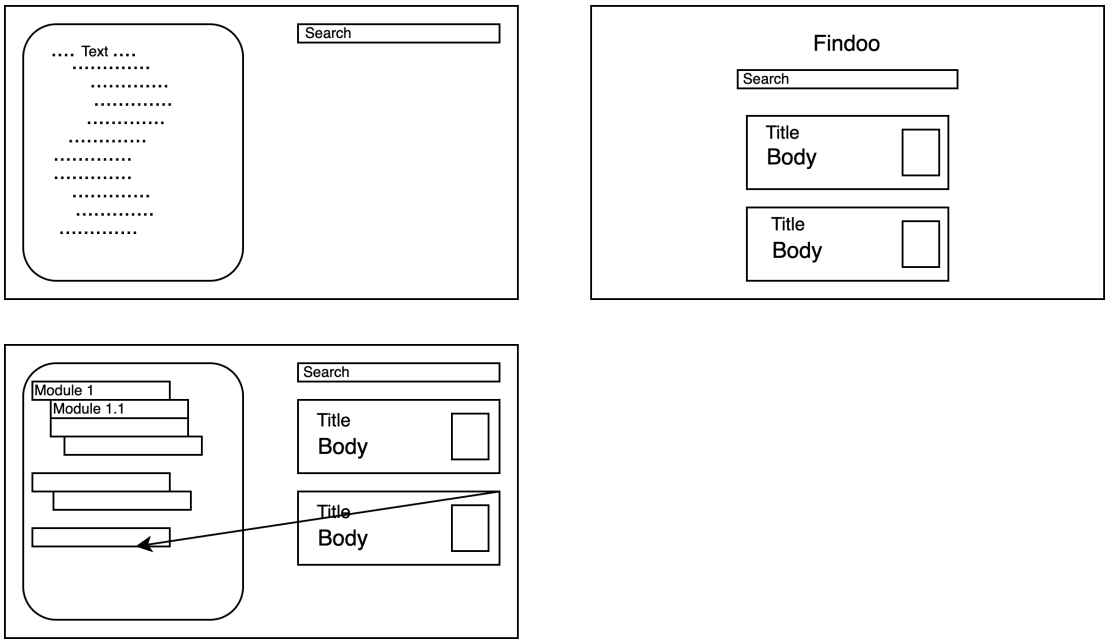


Abbildung 4.2: Das UI-Konzept des Systems. Dies sieht eine Suchmaske, einen textbasierten und einen baumbasierten Editor vor.

und interaktive Inhalte werden nicht explizit (d.h. nicht unabhängig des sichtbaren Textes) unterstützt. Es soll sowohl eine klassische Volltextsuche als auch semantische Suchfunktionalität enthalten. Letztere wird via Natural Language Processing bereitgestellt.

Die Unbekanntheit des Nutzungsszenarios hat auf die syntaktische Volltextsuche zunächst nur wenig Einfluss, sie hat aber eine zentrale Implikation für das System mit Hinblick auf die Natural Language Processing-Komponenten. In klassischen NLP-Projekten ist der Ablauf wie folgt: Zunächst findet die Auswahl eines vortrainierten Modells statt, dieses wird dann auf repräsentativen, spezifischen Inhalten trainiert und anschließend eingesetzt [NOC11]. Hier ist der zweite Schritt der Modell-Verfeinerung nicht möglich. Diese Eigenschaft hat zwei Folgen: Zunächst bedeutet der Wegfall des Trainingsschrittes, dass nur NLP-Modelle verwendet werden sollten, die auch ohne Fine-Tuning gute Ergebnisse liefern. Die zweite Folge ist, dass der zu entwerfende Algorithmus nur eine geringe Bindung an die konkreten Modelle haben darf, um eine leichtes Austauschen dieser zu ermöglichen. Das Austauschen hierbei kann entweder die General-Purpose Performance verbessern (z.B. durch Austausch von BERT [Dev+19] durch RoBERTa [Liu+19], letzteres ist auf einem größeren Korpus trainiert) oder, falls der schlussendliche Anwendungsfall feststeht, durch ein spezifischeres Modell (z.B. BERT durch BioBERT [Lee+19], letzteres ist auf einem medizinischen Korpus trainiert). Die

Abbildung dieser Anforderungen auf die Architektur wird in den nachfolgenden Sektionen erklärt. Technische Details werden in Kapitel 5 gegeben und der Einfluss der Modelle auf die Performance des Systems werden in Kapitel 6 ausführlich diskutiert.

4.2 Robuste Crawling-Strategien

In der Literatur zu Information Retrieval wird der eigentliche Crawling-Vorgang nur selten im Detail betrachtet, da dieser oft nur sehr technische Aspekte beinhaltet. Je nach Anwendungsfall kann es bei diesem aber auch zu konzeptionellen Fragestellungen kommen. In dieser Arbeit betreffen diese Forschungsfragen die Interoperabilität des Systems, die Robustheit des Systems und die Abbildung dieser Anforderungen auf die Softwarearchitektur des Deep Crawlers. An dieser Stelle muss auch festgehalten werden, dass die Geschwindigkeit der Crawling-Komponente nicht kritisch ist, die der Suche allerdings entscheidend. Die zentrale Herausforderung der Crawling-Komponente ist die Erfüllung eines operationellen Robustheitsbegriffes: Das System soll stabil gegenüber kleinen Änderungen sein und einfach an große Änderungen anpassbar sein. Ideal wäre es im Rahmen des Crawlings komplett von der HTML-Struktur des LMS zu abstrahieren. Auf technischer Seite könnte dies via *Content Item Requests* des Standards Learning Tools Interoperability (LTI) [IMS19a] realisiert werden. Allerdings ist dies für viele Learning Management Systeme nicht möglich, dieses Feature wird nicht unterstützt. Eventuell könnte diese Funktionalität in Zukunft via eines Plugins emuliert werden, die technische Umsetzbarkeit ist an dieser Stelle aber unklar und wird nicht weiter verfolgt.

In Abb. 4.3 ist der Deep Crawling-Prozess dargestellt. Dieser läuft ab wie folgt: Zu Beginn werden die Start-Informationen in Form einer Tool ID und einer Provider ID übergeben. Diese identifizieren das LTI-Tool, das durchsucht werden soll und werden im nächsten Schritt vom Launch-Modul verwendet, um das Tool zu öffnen. Zu diesem Zweck wird mit einer API kommuniziert, die die technischen Aspekte wie das Erstellen und Signieren des LTI Basic Launch [IMS19a] implementiert. Dies kann entweder die richtige CLM-API sein oder ein eingebautes Mock-CLM-Modul, dieses wird in späteren Sektionen beschrieben. Ist der Launch erfolgt, wird die URL der Landing Page in das Frontier des Crawlers übertragen und die Crawling-Schleife beginnt. Bei diesen URLs handelt es sich um die Deep Links. In jeder Iteration wird eine URL ausgewählt, in der Regel ist dies die älteste, d.h. die erste in der zeitgeordneten Liste, die das Frontier implementiert. Dann navigiert der Crawler zu dieser URL und klassifiziert den assoziierten Inhalt anhand einer Reihe von Kriterien, Details zu diesem Prozess werden weiter unten erläutert. Wird der Inhalt unterstützt, wird das Inhaltstyp-spezifische Handler-Modul aufgerufen und die Seite gegebenenfalls prä-prozessiert. Dabei kann es sich z.B. um

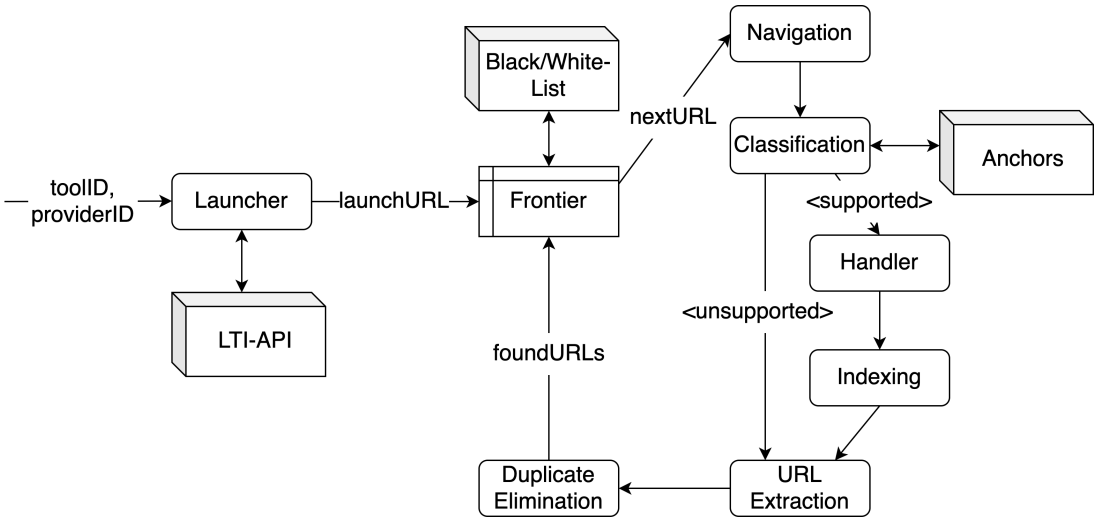


Abbildung 4.3: Der Ablauf des LMS-Crawlers. Ausgehend von einem LTI-Tool werden alle von dort erreichbaren Inhalte durchsucht.

das (Neu-)Starten von Aufgaben und Quizzes oder um die manuelle Navigation in Collection-Datentypen handeln. Je nach Anwendungsfall können hier auch mehrere Indexing-Operationen aufgerufen werden. In jedem Fall wird aber mindestens eine Seite indiziert. Details zu diesem Prozess werden in Sektion 4.3 dargestellt. Nach diesen Schritten (oder falls der Inhalt nicht unterstützt wird) werden die URLs der Seite extrahiert und Duplikate sowie sich bereits im Frontier befindende URLs eliminiert. Die Seite ist nun abgehandelt und eine neue Iteration kann beginnen. Dies wiederholt sich bis das Frontier keine URLs mehr enthält.

Wie oben angesprochen ist die zentrale Herausforderung bei diesem Prozess die Robustheit: Der Algorithmus sollte sowohl konzeptionell als auch (software-)technisch nur minimale Abhängigkeiten von einem bestimmten LMS in einer bestimmten Version haben. Der folgende Abschnitt listet und beschreibt die wichtigsten Charakteristiken und Strategien des Crawlers sowie deren Verwendungskontext und Einfluss auf die Robustheit:

Blacklists Das Extrahieren (die Discovery) von URLs auf einer gegebenen Seite ist ein zentraler Schritt des Crawling-Algorithmus und technisch nicht komplex. Allerdings muss stets ein potentieller Tradeoff mitigiert werden: Vollständigkeit ist eine zentrale Anforderung an jedes Information Retrieval System und um diese sicherzustellen müssen, in Abwesenheit von verfeinernden, a-priori-Informationen, alle erreichbaren Seiten auch analysiert werden. Dies führt allerdings nicht nur zu großem Performance- und Networking-Overhead, sondern auch zu unerwünschten (wenn nicht gar fatalen) Seiteneffekten. Je nach freigegebener Funktionalität

und LMS-Struktur ist es beispielsweise möglich durch zielloses, beliebiges Besuchen von URLs E-Mails an alle Mitglieder zu senden, Foren-Beiträge zu erstellen, oder den Nutzer vom Kurs abzumelden. Zusammenfassend gilt also: Im Allgemeinen sind nur wenige URLs relevant und viele URLs müssen aktiv vermieden werden. Eine Methode den Crawler diesbezüglich abzusichern ist eine Blacklist, also eine Menge von URLs, die, falls entdeckt, aus dem Frontier entfernt werden müssen bevor sie besucht werden. Diese muss folgende Anforderungen erfüllen: Sie muss parametrisierbar sein, um später an sich ändernde LMS-Eigenschaften anpassbar zu sein und Prefix Matching erlauben, da es nicht möglich ist mit vertretbarem Aufwand alle URLs mit allen potentiellen Argumenten vollständig zu spezifizieren.

Whitelists Bei Whitelists handelt es sich um einen zu Blacklists komplementären (und strengerern) Ansatz. Es ist dem Crawler nur erlaubt URLs zu besuchen, die sich in der Whitelist befinden, bzw. deren Präfix in der Whitelist enthalten ist. Dies ist ein sehr sicherer Ansatz, kann aber natürlich den Upfront-Aufwand erhöhen oder die Vollständigkeit senken. Deshalb ist er in der Regel mit anderen Mechanismen (insb. der Nutzung architektureller Invarianten) zu kombinieren.

Nutzung architektureller Invarianten Im Allgemeinen ist die Struktur eines Learning Management Systems nicht unbekannt oder vollkommen beliebig und kann somit im Crawling-Algorithmus berücksichtigt werden. Dies hat allerdings die nachteilige Eigenschaft, dass das Crawling in zukünftigen Versionen oder anderen Konfigurationen des Systems nicht mehr funktioniert - ein klarer Widerspruch zu der Zielsetzung dieses Systems. Die Ausnutzung bestimmter LMS-Eigenschaften ist also nur sinnvoll, wenn diese invariant und architektureller Natur sind. Mit anderen Worten: Werden diese geändert, ist das LMS ein anderes System, nicht nur eine modifizierte Version. Beispiele für solche Eigenschaften sind die direkten, *goto*-URLs in Ilias und die xBlock-Adressen von Open edX. xBlocks erlauben das direkte Springen zu gegebenen Inhalten ohne Navigation und sind die eigentlichen Entitäten, die via LTI freigegeben werden [Inc21]. Diese können somit vom Algorithmus berücksichtigt werden, ohne dessen Robustheit zu gefährden.

Klassifikation Um die Abdeckung aller relevanten Inhalte sicherzustellen, werden (kontrolliert durch die Black-/Whitelist) zunächst alle gefundenen URLs besucht. Indiziert werden aber nur bestimmte Inhaltstypen. Diese werden LMS-übergreifend vom Crawler bzw. Index selbst definiert und abstrahieren die Vielzahl der LMS-spezifischen Typen, so werden zum Beispiel Moodle-Sequenzen und Moodle-Bücher gemeinsam mit Ilias-Wikis dem Typ *Collection* zugeordnet. Konkret werden so die Typen *Page*,*Collection*, *Quiz*, *Exercise* und *File* definiert.

Ob eine gegebene URL indiziert werden soll, wird im Klassifikationsschritt entschieden. Jeder Typ hat eine Menge von Ankerpunkten - Kriterien, die unterstützte Inhalte charakterisieren. Dabei kann es sich z.B. um Keywords in der URL, um CSS-Klassen oder um bestimmte HTML-Attribute handeln. Für jeden Inhaltstyp werden aus den Ankerpunkten abgeleitete Prädikate berechnet, in der Regel einfach das Auftreten des Keywords. Ausgabe der Klassifikation ist der Typ mit den meisten erfüllten Prädikaten. Werden keine Prädikate erfüllt, wird kein Typ erkannt und die Seite nicht indiziert. In jedem Fall werden aber die URLs extrahiert.

Near-Duplicate Detection Da die LMS-Struktur (bis auf eventuelle Invarianten) unbekannt ist, werden Inhalte im Allgemeinen mehrfach besucht. Wenn URLs aufgrund optionaler HTTP-Argumente aber nicht komplett identisch sind ist nicht statisch vorhersehbar, dass es sich um ein Duplikat handelt. Das Verhindern des mehrfachen Indizieren von Inhalten muss also auf Text-Ebene stattfinden. Wie in Sektion 4.3 angesprochen, wird im Indizierungsschritt der *Body* der Seite bestimmt und indiziert. Diese Bestimmung ist aber nicht deterministisch, es kann also (leicht) verschiedene Bodies des gleichen Inhalt geben. Außerdem kann es, je nach LMS, durchaus sein, dass bestimmte Texte nur manchmal sichtbar sind, z.B. Hinweise zu Aufgaben etc. Es muss also erkannt werden ob Texte *fast* identisch zu bereits indizierten Inhalten sind. Dies geschieht via einer passenden Hash-Funktion und Fingerprinting, Details dazu sind in Sektion 5.2 gegeben.

CSS-Selektoren CSS-Aspekte zählen zu den instabilsten Teilen einer Seite und sind somit keine robuste Grundlage für den Crawler. Es gibt aber auch hier robuste Ansätze und best practices: Falls möglich sollte auf die Verwendung von exakten Klassen oder Attributen verzichtet werden, Teil-Strings oder Präfixe sind vorzuziehen. Bestenfalls sollte die Selektion nur auf Element-Ebene (Paragraph, Heading, Form etc.) erfolgen und sich auf sichtbaren Text beziehen. Dieser ist gegebenenfalls stabiler und später einfach zu modifizieren falls es doch zu großen Änderungen auf der LMS-Seite gekommen ist.

Robuste Crawling-Umgebung In vielen Fällen werden die eigentlichen Inhalte einer Website nicht klassisch alle auf einmal geladen, sondern *lazy*, z.B. bei endlosem Scrollen oder durch asynchrone Ajax-Calls. Um damit umgehen zu können, findet das Crawling in einem tatsächlichen Browser statt: Chromium im headless-Modus.

Visible Text Selectors Aufbauend auf den Überlegungen zu der Verwendung von CSS-Selektoren könnte auch ganz auf die Verwendung von CSS bzw. HTML-Konzepten verzichtet werden. Durch die Übergabe eines Paar von nichtannotierten Texten können auch Regionen

von Webseiten delimitiert werden („Selektion aller Texte zwisschen Keywords A und B“). Eine Erweiterung durch abstraktere Typen wie Listen ist auch möglich. Dieser Ansatz hat aber den Nachteil, dass eine Selektor-Engine und eine solche Anfrage-Sprache implementiert werden müssen, Details dazu werden ebenfalls in Kapitel 5 dargestellt.

4.3 Design eines durchsuchbaren Indexes

Der Index Um die oben genannten Anforderungen zu erfüllen, werden für jede Seite die folgenden 11 Attribute gespeichert:

- Der Titel des Inhaltes: Dieser wird für die Darstellung im Editor, der Suchergebnisse und die Autovervollständigung genutzt.
- Der Typ des Inhaltes: Dieser besteht aus dem LMS-Typ und dem Content-Typ und wird für die Darstellung der Ergebnisse verwendet. Ebenso kann nach diesem gefiltert werden.
- Die Sprache des Inhalts: Bei der Suche kann nach dieser gefiltert werden, ebenso ist zu beachten, dass die semantische Suche nur für englischsprachige Inhalte funktioniert.
- Der Body des Inhaltes: Der eigentliche, natürlichsprachliche Inhalt der Seite, der für die Suche genutzt wird. Dieser wird nach bzw. während des Indizierens via einer Text Ingesting Pipeline analysiert und bearbeitet, für Details sei auf Kapitel 5 verwiesen.
- Der HTML-Inhalt der Seite: Der Quelltext der Seite im *raw*-Format. Dieser kann zukünftig für erweiterte Suchen oder Analysen verwendet werden.
- Die Crawling-Zeit: Der Zeitpunkt des Einpflegens des Inhaltes in den Index. Nach diesem kann gefiltert werden, ebenso kann er für ein Garbage Collection-System verwendet werden, das alte Inhalte löscht.
- Die Konzepte des Inhaltes: Mit dem Inhalt assoziierte Konzepte, diese werden für die semantische Suche verwendet.
- Das Embedding des Inhaltes: Von einem Modell berechneter Einbettungs-Vektor des Bodies, dieser soll dessen Semantik repräsentieren und wird für die semantische Suche verwendet.
- Die URI des Screenshots: Mit jedem Index-Eintrag ist ein Bild (in der Regel ein Screenshot) assoziiert, dieser wird vom Backend gehostet und kann via dieser URI abgerufen werden.
- Die URL des Inhaltes: Die konkrete URL, an der der Inhalt gefunden wurde. Im Rahmen des Editors wird diese benötigt, um den assoziierten Inhalt direkt zu öffnen.
- Die Tool-ID und Provider-ID: Diese zwei IDs identifizieren das LTI-Tool [IMS19a], in dem der Inhalt gefunden wurde. Sie werden für die Abbildung auf den Common Cartridge-Standard [IMS15] benötigt.

Der Indizierungsalgorithmus Das Berechnen der im Index gespeicherten Features geschieht im Indizierungsalgorithmus, der in Abbildung 4.4 dargestellt ist. An dieser Stelle werden viele Details nicht diskutiert, diese werden stattdessen zentral in Kapitel 5 gegeben. Die Eingabe des Algorithmus ist ein HTML-Dokument mit assoziierter URL. Der Algorithmus muss nun herausfinden ob diese einen unterstützten E-Learning-Inhalt enthält und von welchem Typ dieser ist. Ist diese Klassifikation erfolgt, müssen die für die Suche erforderlichen Features abgeleitet werden. Wie die Abbildung andeutet, ist der Algorithmus in Stufen und Subprozesse unterteilbar, diese sind orthogonal zueinander und können ggf. parallelisiert werden. Der konkrete Ablauf ist wie folgt: Zunächst wird das Input-Dokument klassifiziert. Dies erfolgt anhand von Ankerpunkten. Dabei handelt es sich um Attribut-Wert-Paare, die typischerweise (aber nicht unbedingt immer) mit einem Inhaltstyp eines LMS assoziiert sind. Die Ausgabe der Klassifikation ist der Inhaltstyp mit den meisten erfüllten Ankerpunkten oder der generellere Typ bei gleichen Anzahlen. Sind keine Ankerpunkte erfüllt wird der Inhalt nicht unterstützt und nicht indiziert. Bei der Klassifikation werden außerdem Meta-Daten wie der Entdeckungszeitpunkt, die URL der Seite und LTI-Tool-Informationen gespeichert. Nach erfolgreicher Klassifikation wird die Find-Body-Heuristik angewendet, um den eigentlichen Inhalt der Seite zu finden, d.h. den relevanten Text der Seite ohne Markup: den *Body*. Ist dieser identifiziert, werden drei voneinander unabhängige Subprozesse gestartet. Der *Concept Extraction*-Prozess identifiziert die im Text vorkommenden Konzepte der verwendeten Ontologie. Dies funktioniert in drei Teilschritten (für Details siehe Kapitel 5 und [Men+11; Dai+13]):

1. *Spotting*: Erkennung der Satzteile (Phrasen), die möglicherweise ein Konzept der Ontologie enthalten.
2. *Candidate Mapping*: Abbildung der Phrasen auf konzeptidentifizierende Kandidaten.
3. *Disambiguation*: Auflösung von Mehrdeutigkeiten durch Betrachten des Kontext um die Phrase.

Der Ablauf ist zusätzlich abhängig von Konfigurationsparametern wie Confidence-Thresholds für Kandidaten. Die extrahierten Konzepte werden dann im Index als ungeordnete Liste gespeichert. Der zweite Subprozess ist das Berechnen einer semantischen Einbettung des Textes, also die Repräsentation des Inhaltes als ein hochdimensionaler Vektor unter der Bedingung, dass Distanzmetriken zwischen mehreren solcher Vektoren dann die semantische Ähnlichkeit der jeweiligen Dokumente hinreichend abbilden. Die Konstruktion dieser Vektoren besteht aus drei Schritten:

1. *Sentence Segmentation*: Zunächst wird das Dokument in Sätze zerlegt, diese sind in der Regel durch Satzzeichen, syntaktische Abhängigkeiten etc. unterscheidbar. Stichwortartige Inhalte werden zwischen angrenzenden Sätzen zu einem Satz zusammengefasst.

2. *Embedding*: Die Einbettung erfolgt satzweise via eines trainierten Modells. Für n Sätze besteht der Zustand des Algorithmus nun aus n' Vektoren der Dimension $m' \times 1$, m' ist hierbei vom konkret verwendeten Modell abhängig. Abhängig von der Einbettungsstrategie kann $n' > n$ gelten, z.B. durch tokenweises Einbetten.
3. *Aggregation*: Die n' Vektoren werden nun aggregiert und auf einen Vektor der Dimension $m \times 1$ abgebildet, m muss indexweit konstant sein. In der Regel wird $m = m'$ gelten, z.B. durch die Aggregationsfunktion des komponentenweisen Durchschnitts.

Als dritte Subfunktion wird der Body selbst indiziert, dies geschieht durch eine Ingestion- und Analyse-Pipeline. Dabei werden eine Reihe von Operationen durchgeführt und vorberechnet, unter anderem wird der Text tokenisiert, Stemming durchgeführt, illegale Zeichen gefiltert und für die Suche notwendige Werte wie Termfrequenzen berechnet, Details sind zum Beispiel Gormley und Tong [GT15] zu entnehmen.

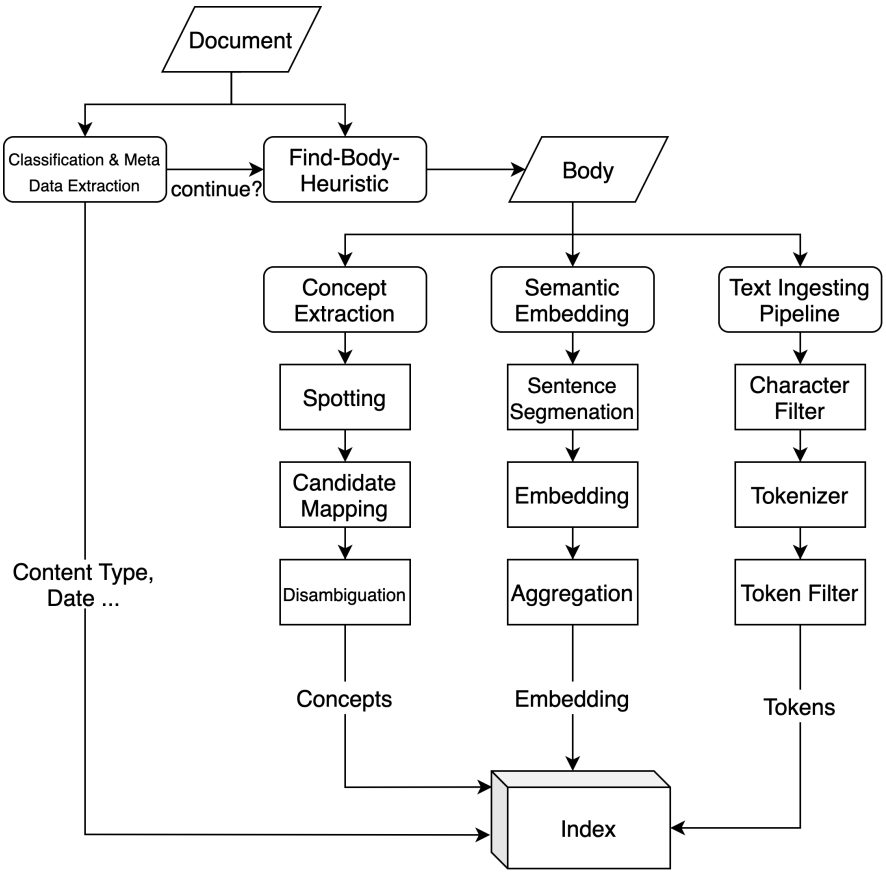


Abbildung 4.4: Das Indizieren eines Dokuments. Dies läuft in drei orthogonalen Subprozessen ab, die die für die drei Suchmethoden nötigen Features berechnen.

4.4 Entwurf des Suchalgorithmus

Diese Sektion beschreibt aufbauend auf den vorhergegangenen Sektionen das Design einer Familie von Suchalgorithmen, die die erwähnten Anforderungen der Domänenunabhängigkeit und Erweiter- bzw. Modifizierbarkeit erfüllen. Sie bestehen alle aus zwei Arten von Komponenten: Implementierungen von syntaktischer Suche auf der einen und semantischer Suche auf der anderen. Als Eingabe erwarten sie immer eine Zeichenkette von einem oder mehreren Keywords. Generell ist zu beachten, dass a-priori keinesfalls eindeutig ist welche Architekturen und Methoden offensichtlich immer überlegen sind. Ein Grund hierfür ist die hohe Zahl der Freiheitsgrade. Zum Beispiel gibt es, falls eine Einbettungs-Komponente verwendet wird, sehr viele Design-Entscheidungen zu treffen: Unter anderem Fragen nach der Einbettungs-Methode, des Modells, welche Teile des Dokuments eingebettet werden sollen, dem Einfluss von Präprozessierung oder eine eventuelle Kombination von Techniken. Fragestellungen dieser Art werden in Kapitel 5 ausführlicher diskutiert. Außerdem ist die Bewertung der Performance nicht offensichtlich. So ist zum Beispiel die relative Präferenz von hohem Recall oder hoher Präzision nutzerabhängig und das Kriterium der *Serendipity*, also des zufälligen Finden von Inhalten, deren Relevanz im Vorhinein nicht abschätzbar war, lässt sich oft gar nicht messen.

Konkret werden im Folgenden zwei Modelle vorgestellt: Ein Minimal-Modell, dass die besagten Komponenten verwendet und eine mögliche Erweiterung. Wie sich in Kapitel 6 zeigen wird, ist deren Performance aber annähernd äquivalent. Ein weiteres Feature des hier entworfenen Information Retrieval Systems ist, dass die Formeln naheliegend auf ein Recommender-System angewendet werden können. Dies wird in Sektion 4.5 dargestellt.

Intuition Eine naheliegende und sehr relevante Methode des Suchens ist streng syntaktisch vorzugehen. Kommt ein Keyword in einem Dokument vor, soll dieses auch wiedergegeben werden. Die zentrale Aufgabe des Algorithmus ist nun die des Ranking, also das Sortieren von Dokumenten mit einem oder mehreren Vorkommnissen des Keywords nach deren relativen Relevanz zu dem Informationsbedürfnis des Nutzers, das durch das Keyword (in der Regel suboptimal) ausgedrückt wird. Zudem ist die Definition eines Vorkommnis nicht eindeutig: Kommt beispielsweise nur ein Teilwort oder eine andere grammatikalische Form des Wortes vor, so muss dies auch berücksichtigt werden, wenn auch das Dokument dann ggf. weniger relevant ist als es ein direktes Vorkommen ausdrücken würde. In dieser Arbeit wird dieses *syntaktische Matching* durch den bekannten *Term Frequency - Inverse Document Frequency* (TFIDF)-Algorithmus [Ram+03] erzielt. Auf technischer Seite funktioniert dies u.a. über das *Stemming*, also das Reduzieren von Keywords und Index-Einträgen auf ihre Grundform. Details dieser Art werden allerdings von bestehenden Implementierungen übernommen.

In vielen Fällen sind für ein bestimmtes Informationsbedürfnis auch Dokumente relevant, die die gewählten Keywords nicht enthalten oder, auf der anderen Seite, zeigt das Vorkommen eines Keywords nicht unbedingt die Relevanz des Dokumentes an. Das Auffinden bzw. Ausschließen dieser Dokumente ist die Aufgabe der *semantischen Suche*. In dieser Arbeit werden zwei Methoden verwendet: Die erste ist das Berechnen der Distanz der semantischen Einbettungen von Dokumenten und Keywords - ist die Einbettung aussagekräftig, so kann die Vektor-Distanz zweier Einbettungen als Maß für die inhaltliche Ähnlichkeit und somit die Relevanz des Dokuments bezüglich des Keywords gesehen werden. Die Eigenschaft, dass semantische Einbettungen vergleichbar sind ist nicht selbstverständlich. Reimers et al. [RG19] erläutern, dass selbst state-of-the-art Performance in NLP-Aufgaben nicht ohne weiteres auf semantische Suchen erweitert werden kann, hierfür werden spezielle Modelle benötigt. Die zweite Methode ist die Berechnung von *Shared Concepts* von Keywords und Dokumenten bezüglich einer Ontologie. Jedes Dokument wird mit den Konzepten einer Ontologie annotiert, die in diesem vorkommen. Werden nun für die gegebenen Keywords ebenfalls relevante Konzepte hergeleitet (ein nichttrivialer Prozess), kann wieder eine Relevanz-Metrik definiert werden: Je mehr Konzepte ein Dokument und die Keywords gemeinsam haben, desto relevanter ist das Dokument.

Der Ablauf des Algorithmus Die Ideen des vorherigen Abschnitts werden in dieser Arbeit im Algorithmus der *Ensemble Search* kombiniert. Dieser ist in Abbildung 4.5 dargestellt. Es existieren Varianten mit einem oder zwei Schritten und es ist wieder nicht a-priori bekannt welche zu besseren Ergebnissen führt: Das einschrittige Modell bezieht die Ontologie-Informationen sofort mit ein, hat aber das Problem, dass Konzepte aus den nur wenigen Keywords abgeleitet werden müssen. Basierend auf den Query-Keywords werden die Ontology-, TFIDF- und Embedding-Scores berechnet und gewichtet zum Ensemble Score aufsummiert. Die Ausgabe ist dann ein Ranking aller Dokumente gemäß diesen Scores. Dem anderen Modell liegt die folgende Annahme zugrunde: Die ersten k Ergebnisse werden ohne Ontologie-Informationen berechnet und sind trotzdem relevant genug, sodass die Vereinigung deren Konzepte eine gute Grundlage für ein anschließendes Re-Ranking ist. Hier werden zunächst nur der TFIDF- und der Embedding-Score wie oben berechnet und gewichtet aufsummiert. Dann werden die Konzepte der relevantesten k Dokumente als eine Menge betrachtet und ontologisch erweitert. Das heißt, dass der Relevanz-Score aller Dokumente um den Ontologie-Score erweitert wird. Dieser wird nun nicht auf Basis der Query-Konzepte berechnet, sondern auf Basis dieser neuen Menge der Common Concepts. Dies vermeidet das komplexe Ableiten von Konzepten auf Basis der Query-Keywords. Die Ausgabe des Algorithmus ist dann dieses Re-Ranking.

Die fundamentalen Annahmen für die entworfenen Modelle sind:

- Für lange Texte sind extrahierte Ontologie-Konzepte ein gutes Maß für deren Inhalt, da viel Kontext-Information gegeben ist.
- Für kurze Texte sind semantische Einbettungen ein gutes Inhalts-Maß, da diese in der Regel von den aussagekräftigen Worten dominiert werden. Dieser Effekt tritt allerdings auch bei längeren Texten auf.
- TFIDF ist stets anwendbar und stabil, aber inhärent auf die Syntax limitiert.

Ensemble-Score Gleichung 4.1 gibt den Score eines Dokuments d_i bezüglich einer Query q an. Wie im vorhergegangenen Abschnitt beschrieben, besteht dieser aus den drei Komponenten des TFIDF-Score ts, des Ontology Score os und des Embedding Score es, die nachfolgend genauer erläutert werden. Der Beitrag dieser Komponenten zum Gesamt-Score ist mit den Gewichten w_1, w_2, w_3 parametrisiert. Diese können dem Algorithmus als zusätzlichen Input übergeben werden, sonst sind sie mit (konservativen) Standard-Werten belegt.

$$\text{score}(q,d_i) = w_1 \cdot \text{ts}'(q,d_i) + w_2 \cdot \text{os}'(q,d_i) + w_3 \cdot \text{es}'(q,d_i) \tag{4.1}$$

Bei den Komponenten handelt es sich jeweils um auf das Intervall [0,1] normalisierte Versionen der unten beschriebenen Funktionen. Die (Pseudo-)Normalisierung findet gemäß Gleichung 4.2 statt und ist insbesondere für den theoretisch unbeschränkten TFIDIF-Score relevant.

$$s(q,d_i)' = \begin{cases} 1 & , \; s(q,d_i) > Q_s(0.9) \\ 0 & , \; s(q,d_i) < Q_s(0.1) \\ \frac{s(q,d_i) - Q_s(0.1)}{Q_s(0.9) - Q_s(0.1)} & , \; \text{sonst} \end{cases} \tag{4.2}$$

Hierbei ist s eine der Score-Komponenten und $Q_s(0.9)$ bzw. $Q_s(0.1)$ geben das neunte bzw. das erste Dezil des Wertebereiches von s auf dem unterliegenden Datensatz an. Diese müssen also vorher bekannt sein bzw. abgeschätzt werden. In der eigentlichen Implementierung werden die auf dem Testdatensatz bestimmten Dezil-Werte verwendet, die hinreichend repräsentativ für die meisten Anwendungsfälle sind, ein Anpassen dieser Parameter ist aber möglich.

TFIDF-Score Der TFIDF Score entspricht in seiner Formulierung der Implementierung aus Elasticsearch [DG13; GT15] und ist in Gleichung 4.3 dargestellt.

$$\text{ts}(q,d) = \text{querynorm}(q) \cdot \text{coord}(q,d) \cdot \sum_{t \in q} \text{tf}(t,d) \cdot \text{idf}(t)^2 \cdot \text{boost}(t) \cdot \text{fieldnorm}(t,q) \tag{4.3}$$

Er besteht aus den folgenden Komponenten [GT15]:

- Der *query normalization factor* $\text{querynorm}(q) = \frac{1}{\sqrt{\sum_{t \in q} \text{idf}(t)^2}}$ dient dazu die Scores verschiedener Queries vergleichbar zu machen, der Effekt ist aber minimal und hat keinen Einfluss auf das eigentliche Ranking.
- Der *coordination factor* $\text{coord}(q,d) = \frac{|\{t \in q \cap d\}|}{|\{t \in q\}|}$ belohnt Dokumente, die einen prozentual hohen Anteil der Query-Terme enthalten.
- Die Termfrequenz $\text{tf}(t,d) = \sqrt{\text{freq}(t,d)}$ misst die Zahl der Vorkommen (die Frequenz) eines Query-Terms in einem Dokument.
- Die inverse Dokumentfrequenz $\text{idf}(t) = 1 + \log \frac{|D|}{|\{d \in D | t \in d\}|+1}$ betont Terme, die im Textkorpus selten vorkommen.
- Der Boost-Term $\text{boost}(t)$ ist ein Faktor, der als Parameter übergeben werden kann, um den Score eines Terms manuell zu erhöhen.
- Die *field length norm* $\text{fieldnorm} = \frac{1}{\sqrt{|t \in \text{d.f}|}}$ ist die Inverse Quadratwurzel der Zahl der Terme in Feld f von d.

Ontology-Score Gleichung 4.4 gibt den Ontology-Score an, dieser entspricht dem Jaccard-Index [LW71] der Menge aller abgeleiteten Konzepte der Query q und der Menge der annotierenden Konzepte des Dokuments d bzw. den annotierenden Konzepten der beiden Dokumente im zweischrittigen Algorithmus. Für zwei Mengen S_1,S_2 ist der Jaccard-Index J definiert als Quotient der Kardinalitäten der Schnittmenge und der Vereinigung: $J(S_1,S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$. Er ist als solcher auf den Wertebereich [0,1] normalisiert. Die Aussagekräftigkeit des Ontology Scores ist stark von den Eigenschaften der verwendeten Ontologie abhängig. Insbesondere deren Tiefe ist wichtig. In dem hier vorliegenden Anwendungsfall wird die DBpedia-Ontologie verwendet. Als General-Purpose-Ontologie ist diese sehr *shallow*, sie enthält 4.23 Millionen Einträge, verteilt auf 685 Klassen und 2795 Attributen [Leh+15]. Da eine Anforderung an den Suchalgorithmus die Domänenunabhängigkeit ist, ist sie sehr gut geeignet, da für fast alle Inhalte Konzepte gefunden werden können. Für spezifische Anwendungsfälle ist sie allerdings weniger gut geeignet: Wenn nahezu alle Dokumente mit den gleichen Konzepten annotiert sind, sinkt die Bedeutung der Shared-Concept-Metrik stark. Dieses Phänomen wird in Kapitel 6 illustriert.

$$os(C_q, C_d) = J(C_q,C_d) = \frac{|C_q \cap C_d|}{|C_q \cup C_d|} \tag{4.4}$$

Die Verwendung des Jaccard-Index (anstatt der einfachen Kardinalität der Schnittmenge zum Beispiel) hat zwei Gründe: Erstens ist dieser per Definition auf [0,1] normalisiert, zweitens kann dieser mit stark unterschiedlichen Kardinalitäten umgehen. Die Mengen C_q, C_d sind nämlich oft sehr unterschiedlich groß: Bei manchen Keywords lassen sich nur wenige Konzepte ableiten und $|C_q|$ ist sehr klein. Dies tritt vor allem beim einschrittigen Ensemble-Algorithmus

auf. An dieser Stelle wird eine statische, vom System vorhergesehene Ontologie verwendet. Als mögliche Erweiterung könnte wie bei einigen Systemen in Kapitel 2 auch parametrisiert vorgegangen werden, indem der Nutzer eine ggf. spezialisiertere Ontologie mit übergibt.

Embedding-Score Gleichung 4.5 definiert den Embedding-Score. Dieser ist gegeben durch die Standardmetrik der Cosinus-Ähnlichkeit [MRS08] der n-dimensionalen Vektoren der semantischen Einbettung der Query v^q und dem des Dokuments v^d.

$$\text{es}(v^q, v^d) = \frac{v^q \cdot v^d}{\|v^q\| \|v^d\|} = \frac{\sum_{i=0}^{n} v_i^q v_i^d}{\sqrt{\sum_{i=1}^{n} (v_i^q)^2} \sqrt{\sum_{i=1}^{n} (v_i^d)^2}} \tag{4.5}$$

Formell ist die Cosinus-Ähnlichkeit auf das Intervall $[-1,1]$ normiert: -1 für genaue Gegenteiligkeit, 0 für Unabhängigkeit und 1 für Identität. Bei semantischen Einbettung ist diese aber *skewed* auf ungefähr das Intervall $(-0.2, 1]$. Der Wert 1 ist mit einem Selbstvergleich trivial zu erreichen, aber genaue Gegenteiligkeit tritt nie auf: Selbst eigentliche Gegenteile wie *groß* und *klein* sind semantisch verwandt - beides sind Adjektive, beschreiben pyhsische Eigenschaften etc. Die *Black-Box*-Eigenschaft des hier betrachteten generischen Modells und NLP-Modellen generell hat außerdem zur Folge, dass einzelne Komponenten der Einbettungsvektoren keine Semantik haben und fortgeschrittenere Analysen derselben nicht ohne weiteres möglich sind. Die konkrete Einbettung eines Dokuments ist der satzweise Durchschnitt der Einbettungen aller seiner Sätze, siehe Kapitel 5.

Alternativen Wie oben angedeutet sind die Komponenten der syntaktischen und semantischen Suche prinzipiell frei kombinierbar. In der Literatur, z.B. in Frei et al. [FSC20], werden oft Stichworte extrahiert und diese eingebettet, hier wäre dies analog zum Einbetten der ontologischen Konzepte. Dies kann in Zukunft an anderer Stelle weiter verfolgt werden, erste Experimente zeigen aber keine bessere Performance. Dies hat wahrscheinlich mit dem Einbettungs-Modell zu tun. Ist dieses oder die Ontologie, wie hier, General-Purpose sind spezifischere Einbettungen nicht aussagekräftiger. Eine mögliche Erklärung wäre, dass die spezifischeren Worte im Trainings-Korpus des Modells seltener auftreten und dementsprechend die Einbettung derselben weniger akkurat ist, was wiederum eventuelle Vorteile des gezielten Einbetten von repräsentativeren Worten negiert.

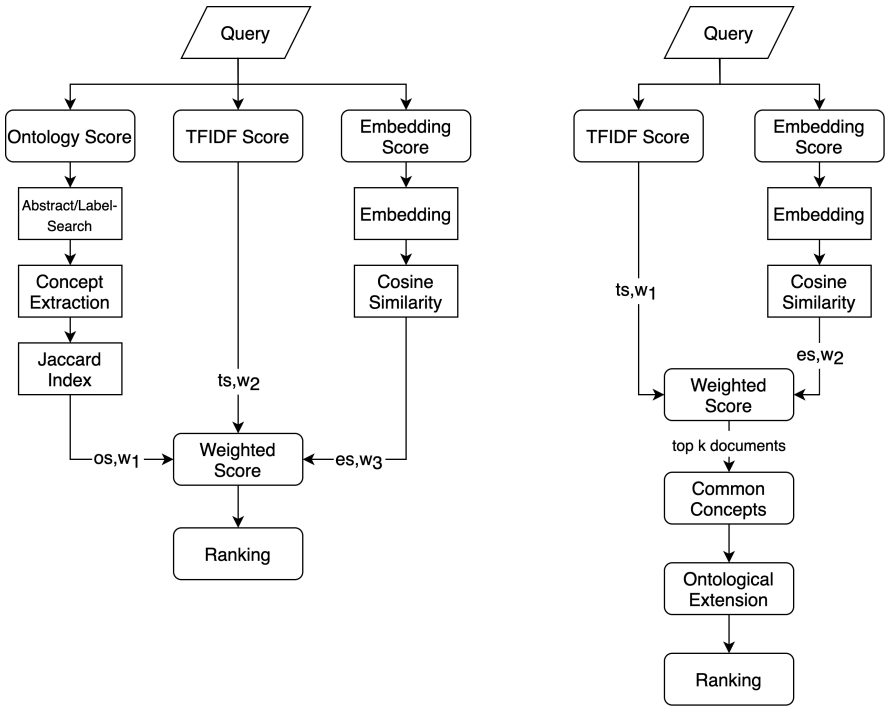


Abbildung 4.5: Der Ablauf der Ensemble Search mit einem oder zwei Schritten. Im ersten Algorithmus müssen Konzepte für die Nutzung der Ontologie basierend auf der Query hergeleitet werden, der zweite umgeht dieses Problem durch ein Re-Ranking.

4.5 Integration eines Recommender-Systems

Die Felder von Information Retrieval und Recommender-Systemen sind verwandt, werden in der Literatur aber oft strikt getrennt betrachtet, wie auch bei den möglichen Erweiterungen am Ende dieser Sektion. Im Rahmen des implementierten Algorithmus werden aber die Techniken des Such-Algorithmus wiederverwendet. Das Ziel des Recommender-Algorithmus ist das Finden von ähnlichen Inhalten, gegeben eine (im Rahmen eines Kurses) strukturierte Menge von Inhalten. Zu diesem Zweck werden die Formeln 4.4 und 4.5 angewendet. Es ist zu beachten, dass diese aufgrund ihrer Formulierung nicht angepasst werden müssen, denn der Ursprung des Query-Vektors v_q und der Query-Konzepte C_q ist beliebig. Nun werden also nicht die Keywords der Query eingebettet, sondern ganze Lernressourcen, gleiches für die Konzept-Extraktion im Rahmen des Ontology-Scores. Die Aufgabe des Recommender-Systems beschränkt sich also auf die geeignete Aggregation bzw. Transformation der gegebenen Input-Informationen, die Konzepte und Embeddings sind bereits berechnet und im Index gespeichert.

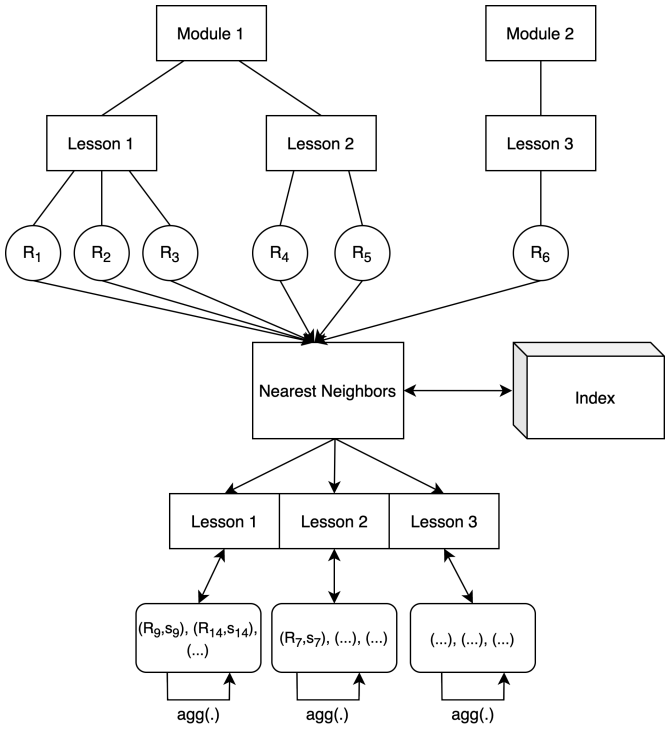


Abbildung 4.6: Der Ablauf des Recommender-Algorithmus: Für alle Ressourcen im aktuellen Kurs werden jeweils die k ähnlichsten Ressourcen bestimmt und anschliessend aggregiert. Die eigentlichen Vorschläge finden dann pro Eltern-Knoten statt.

Der Algorithmus Der grundlegende Ablauf des Recommender-Systems ist in Abbildung 4.6 dargestellt. Dieses funktioniert wie folgt: Der aktuelle Kurs wird als Wald von Bäumen betrachtet. Er besteht aus beliebig vielen, beliebig geschachtelten Modulen. Ein Modul bezeichnet dabei einen Knoten, der Module oder Ressourcen als Kindknoten hat. In der Abbildung sind Wurzelknoten als Module und Knoten in der ersten Ebene als Lessons bezeichnet, dies ist aber nur illustrativ. Konzeptionell wird nicht zwischen verschiedenen Ebenen unterschieden. Nun wird für jedes Modul, das über Ressourcen als direkte Kindknoten verfügt, isoliert Vorschläge berechnet, in der Abbildung also für die Lessons 1-3. Dabei werden zunächst für jede Kind-Ressource die k ähnlichsten Ressourcen bestimmt, indem die jeweiligen Ontology und Embedding-Scores addiert werden. Es wird also der Ensemble-Score aus Gleichung 4.1 ohne den TFDIF-Score wiederverwendet. Ressourcen, die bereits im Kurs vorhanden sind, werden dabei nicht berücksichtigt. Der nächste Schritt besteht dann aus der Aggregation der Ressourcen auf Modul-Ebene. Betrachten wir ein Modul mit m Knoten. Vor dem Aggregationsschritt haben wir also $m \cdot k$ Vorschläge. Sind diese tatsächlich alle verschieden, werden einfach die k Elemente mit dem höchsten Score als Vorschläge übernommen. In der Regel wird aber dieselbe Ressource mehrfach vorgeschlagen. Dies kann nach dem folgenden Prinzip behandelt werden: Wenn eine Ressource mehrfach vorgeschlagen wird, bekommt sie einen Score gemäß Formel 4.6. Sei R die Menge der Ressourcen i, für die r als Vorschlag berechnet wird. Hierbei ist $\mathrm{score_i(r)}$ der Score von r gemäß Ressource i. Die Intuition hinter dieser Funktion ist die folgende: Sie wächst (sinnvollerweise) mit der Anzahl der Vorschläge, lässt aber auch zu, dass eine andere Ressource l, die weniger oft vorgeschlagen wird, aber dafür mit höheren Scores $\mathrm{score_i(l)}$, auch in die Liste der Top-k-Vorschläge aufgenommen wird bzw. werden kann. Die implementierte Version des Recommender-Systems verwendet allerdings nur den Durchschnitt der $\mathrm{score}_i(r)$-Werte.

$$\mathrm{score'(r)} = 1 + \log|\mathrm{R}| \cdot \frac{\sum_{\mathrm{i}\in\mathrm{R}} \mathrm{score_i(r)}}{|\mathrm{R}|} \tag{4.6}$$

Erweiterungen und Alternativen

Der obige Algorithmus ist bewusst minimal gehalten, auch aus Gründen der Kompatibilität und Erweiterbarkeit. Er erfordert nicht mehr Informationen als das Information-Retrieval-System, ist also *self-contained.* Naheliegende Erweiterungen sind z.B. die Betrachtung von Modulen im Kontext (und nicht nur isoliert) oder die Gewichtung von Vorschlägen abhängig von der Ebene im Kurs. Diese Sektion zeigt weitere, auch grundlegend andere Alternativen auf. Viele erfordern allerdings einen anderen Anwedungskontext wie z.B. die Existenz von Informationen über andere Kurse oder den Nutzer.

Ad-hoc-Vorschläge Im Rahmen eines sehr einfache Recommender-Algorithmus kann die Struktur der durchsuchten Kurse ausgenutzt werden: In den vorhergegangenen Sektionen wurden die indizierten Inhalte in Isolation betrachtet, diese liegen aber in der Regel strukturiert in Learning Management Systemen vor, d.h. eingegliedert in Sektionen, Ordner oder ähnliche aggregierende Elemente. Diese Information kann genutzt werden, um nahe Elemente vorzuschlagen: Kinder, Nachbarn o.ä. Dies hat den Vorteil, dass keine semantische Analyse erfolgen muss. Nachteile sind der erhöhte algorithmische Aufwand zum Crawling-Zeitpunkt, das Speichern zusätzlicher Positions-Daten und keinerlei Qualitätsgarantien. Wenn der Kurs sinnvoll strukturiert ist, sind die Ergebnisse gut, falls nicht kann die Relevanz der Vorschläge sehr gering sein.

Association Rule Mining Dieser Ansatz kommt ebenfalls ohne semantische Analyse der Inhalte aus, sondern betrachtet andere Kurse. Der grundlegende Gedanke ist der folgende: Wenn Inhalte a und b in vielen Kursen zusammenstehen und im aktuellen Kurs nur a enthalten ist, ist b ein sinnvoller Vorschlag. Das analytische Bestimmen dieser Zusammenhänge kann über die Konzepte des *supports* und der *confidence* erfolgen, in dieser Formulierung vorgeschlagen von Agrawal et al. [AIS93]. Seien im Folgenden A und B Mengen von E-Learning-Inhalten und C eine Menge von Kursen, die aus diesen Inhalten bestehen. Der Support der Inhaltsmenge A ist dann der Anteil der Kurse, die diese Inhalte enthalten, gegeben in Gleichung 4.7.

$$\text{supp}(A) = \frac{|\{c \in C | A \subseteq c\}|}{|C|} \tag{4.7}$$

Sei weiter $A \Rightarrow B$ eine Association Rule. Deren Confidence ist gegeben via Gleichung 4.8. Sie ist ein Maß für die anteilige Korrektheit der Assoziation - wie oft sind die Inhalte aus B im gleichen Kurs wie die Inhalte aus A?

$$\text{conf}(A \Rightarrow B) = \frac{\text{supp}(A \cup B)}{\text{supp}(A)} \tag{4.8}$$

Mit diesen Werten können nun auf Basis von A Inhalte B genau dann vorgeschlagen werden, wenn $\text{conf}(A \Rightarrow B)$ größer als ein bestimmter Grenzwert ist [AIS93]. Eine Erhöhung dieses Grenzwertes verbessert die Vorschläge, verringert aber auch deren Anzahl.

Collaborative Filtering Ein klassischer Recommender-Algorithmus ist der des Collaborative Filtering, dieser berechnet Vorschläge für einen Nutzer u aufbauend auf Ratings, die er und zu ihm ähnliche Nutzer, seine Nachbarschaft $N(u)$, vergeben haben [Sch+07]. Dies kann in zwei Ausprägungen erfolgen: Die erste Möglichkeit ist nutzerbasiertes Collabora-

tive Filtering. Gleichung 4.9 gibt die Relevanzvorhersage eines Items i für einen Nutzer u an, sim(u,n) ist die Ähnlichkeit zwischen u und einem Nachbarn n, z.B. gegeben durch den Pearson-Korrelationsfaktor. $\bar{r}_u$ ist der Ratingsdurchschnitt von u, $r_{n,i}$ ist das Rating von n für i und $\bar{r}_n$ ist das durchschnittliche Rating von n für alle Items.

$$\text{pred}(u,i) = \bar{r}_u + \frac{\sum_{n \subset N(u)} \text{sim}(u,n) \cdot (r_{n,i} - \bar{r}_n)}{\sum_{n \subset N(u)} \text{sim}(u,n)} \tag{4.9}$$

Ein zentrales Problem dieses Ansatzes ist das Berechnen einer guten Nachbarschaft [Sch+07]. Der zweite Ansatz arbeitet nicht auf der Ähnlichkeit zwischen Nutzern, sondern zwischen Items und kommt somit ohne diese Vorberechnung aus. Dieses itembasierte Collaborative Filtering [Sch+07] ist in Gleichung 4.10 gegeben. $R(u)$ ist die Menge aller Items, die u bewertet hat und sim(i,j) analog zu oben die Ähnlichkeit der Items i und j.

$$\text{pred}(u,i) = \frac{\sum_{j \in R(u)} \text{sim}(i,j) \cdot r_{u,i}}{\sum_{j \in R(u)} \text{sim}(i,j)} \tag{4.10}$$

Wie Schafer et al. [Sch+07] belegen, führt Collaborative Filtering in der Praxis zu guten Ergebnissen und kann vielseitig angepasst werden. Hier ist dessen Anwendung aber nicht trivial, da sowohl Nutzer-Informationen als auch Ähnlichkeitsmetriken für Nutzer und Items gegeben sein müssen.

Explainability Eines der großen Probleme von Ansätzen des Maschinellen Lernens, insb. bei komplexen Modellen, ist deren Erklärbarkeit - wie hängen Input und Output zusammen? Klassischerweise gilt hier das Black Box-Modell, die menschliche Intuition geht verloren und selbst offensichtliche Fehler werden nicht erkannt. Das hier entworfene System kann naheliegend in dieser Richtung optimiert werden. Insbesondere die (gemeinsamen) Ontologie-Konzepte von Inhalten können leicht angezeigt werden, um Vorschläge plausibel zu machen.

Das Concept-Source-Problem Wie bei der Motivation des zweischrittigen Suchalgorithmus angedeutet, ist ein zentrales Problem bei der Verwendung der Ontologie Konzepte aus dem nur via Keywords definierten Informationsbedürfnis herzuleiten. Hier könnte in Zukunft der weitere Kontext betrachtet werden, um diesen Prozess zu erleichtern. Kontext kann hierbei vieles bedeuten: vergangene Suchen, die Suchen anderer Nutzer, demographische Informationen, Informationen über den Verwendungs-Kontext des Systems etc.

Reasoning Eine weitere (und komplexe) Erweiterung betrifft die Anwendung der Ontologie, diese beschränkt sich bisher auf die Schnittmengenbildung von Konzepten. Diese sind aber

nicht isoliert, sondern in einer Hierarchie angeordnet und haben Relationen zueinander. Diese können via eines Reasoning-Prozesses ausgenutzt werden, um Beziehungen zwischen Inhalten herzustellen. Viele Details sind hier aber noch unklar, unter anderem der Zeitpunkt (offline vs. online) und der Umfang des Prozesses. In der aktuellen Version ist DBpedia aufgrund der Shallowness-Eigenschaft zudem nur sehr eingeschränkt dafür geeignet.

5 Implementierung des Assistenzsystems

Nachdem im vorangegangenen Kapitel die konzeptionellen Aspekte des System diskutiert wurden, wird nachfolgend deren (technische) Umsetzung beschrieben. Das System ist konkret in Kooperation mit Teams aus dem Fraunhofer IOSB und einem Industriepartner entstanden, Details dieser Kooperation werden in diesem Kapitel auch genannt. Im Rahmen der Implementierungsbeschreibung werden auch bisher nur kurz oder gar nicht genannte algorithmische Bestandteile erläutert und abschließend die konkreten Ergebnisse der Arbeit vorgestellt. Sektion 5.1 beschreibt die Architektur des Systems auf technischer und Entwurfs-Ebene, Sektion 5.2 beschreibt eine Reihe von zentralen, technischen Herausforderungen und deren Lösung. Sektion 5.3 detailliert die Freiheitsgrade des Designs und beschriebt die Modell-Auswahl quantitativ. Sektion 5.4 fasst die Beobachtungen zur Interoperabilität von Systemen zusammen, die während der Arbeit festgehalten wurden. Schließlich gibt Sektion 5.5 einen Überblick über das Frontend der Applikation.

5.1 Die Architektur im Überblick

Die grundlegende Architektur des Systems mit allen beteiligten Komponenten und den dort verwendeten Technologien ist in Abbildung 5.1 dargestellt. Wie dort zu erkennen ist folgt das System dem Architekturmuster der Microservices, die via einer leichtgewichteten HTTP-REST-API kommunizieren. Dies ist zum einen eine konsequente Erweiterung der Client-Server-Architektur von Findoo-Server und Findoo-UI und folgt den in Sektion 5.4 dargestellten Regeln. Darüber hinaus sind aber die zentralen Vorteile eine intrinsische, logische Trennung von Belangen (*seperation of concerns*) in Module, die voneinander nur über feste Interfaces abhängig sind und die implizite Unterstützung einer Vielzahl von Technologien und deren jeweiligen Ökosystemen. Ein illustrierendes Beispiel hierfür ist die Berechnung der semantischen Einbettungen. Dieses Modul ist in Python via der Pytorch Deep-Learning-Bibliothek implementiert, in Node.js existiert keine äquivalente Implementierung. Die Struktur und Kommunikation via HTTP ist also in einem gewissen Sinn unumgänglich. Im nachfolgenden werden die einzelnen Module diskutiert:

- *Findoo-UI*: Das Interface für die Nutzerinteraktion. Es kommuniziert mit dem Server-

Backend für die Suche, die Auto-Vervollständigung während des Suchprozesses, die Anzeige von Screenshots während der Sucherergebnispräsentation und im Rahmen des Recommender-Systems. Das Editieren von Common Cartridge-Kursspezifikationen findet auch hier statt. Es handelt sich also nicht um einen klassischen *Thin Client*.

- *Findoo-Server*: Das Backend implementiert die API und delegiert die eigentliche Funktionalität an die konkrete Anwendungslogik in den entsprechenden Modulen. Hier werden extensiv Adapter-Klassen eingesetzt, diese isolieren die API-Logik von der Business-Logik. So wird zum Beispiel stets nur der Recommendation-Adapter angesprochen. Ob dieser die Logik selbst implementiert oder selbst wieder delegiert hat keinen Einfluss auf den Rest des Systems. Ein Austausch der unterliegenden Logik bzw. Implementierung ist also einfach möglich. Das selbe gilt für die Ontology-, Embeddings- und Index-Adapter. Für die Kommunikation werden neben Adaptern auch Daten-Transfer-Objekte (DTOs) eingesetzt, die interne Objekte serialisierbar darstellen und auch als eine Art Interface agieren.
- *Index*: Der Index des Systems entspricht einem Elasticsearch-Index, dieser ist Teil eines eigenen Services, mit diesem wird nur via einer HTTP-API kommuniziert. Neben der inhärenten Effizienz hat Elasticsearch, bedingt durch die Unterstützung von Apache Lucene, den Vorteil, dass dort eine Reihe von state-of-the-art Implementierungen des TFIDF-Suchalgorithmus [Ram+03], der Cosinus-Ähnlichkeit etc. vorhanden sind und nativ Query-Strings unterstützt werden.
- *Embeddings*: Der Embeddings-Service ist mittels der Sentence-Transformers-Bibliothek implementiert und wird mittels Pytorch vektorisiert verwendet, um maximale Performance sicherzustellen. Er wird von einer Flask-Instanz als Server gehostet.
- *Crawler*: Für den Deep Crawler wird Puppeteer verwendet, dabei handelt es sich um eine headless Version von Chromium, der Open Source Variante von Google Chrome. Dieser kann out of the box mit Ajax-Calls und Javascript umgehen, das ist bei anderen Crawlern nicht der Fall. Die Kommunikation mit dem Rest des Systems geschieht auch über Adapter-Klassen, insbesondere den Index-Adapter und einen Session-Launcher für die Durchführung von LTI-Launches [IMS19a] und die Kommunikation mit der CLM-API [IOS21].
- *Provider*: Die Provider sind alle angeschlossenen LTI-Tools. Dabei handelt es sich in der Regel um Learning Management Systeme, theoretisch können aber beliebige Anwendungen LTI-Launches akzeptieren und als Tool eingebunden werden, diese sind jedoch nicht Teil des Systems und stehen nicht unter dessen Kontrolle.
- *CLM*: Die Informationen für die LTI-Launches (insbesondere die Oauth1-Keys, Secrets und Endpoints) werden in der Common Learning Middleware (CLM) verwaltet und

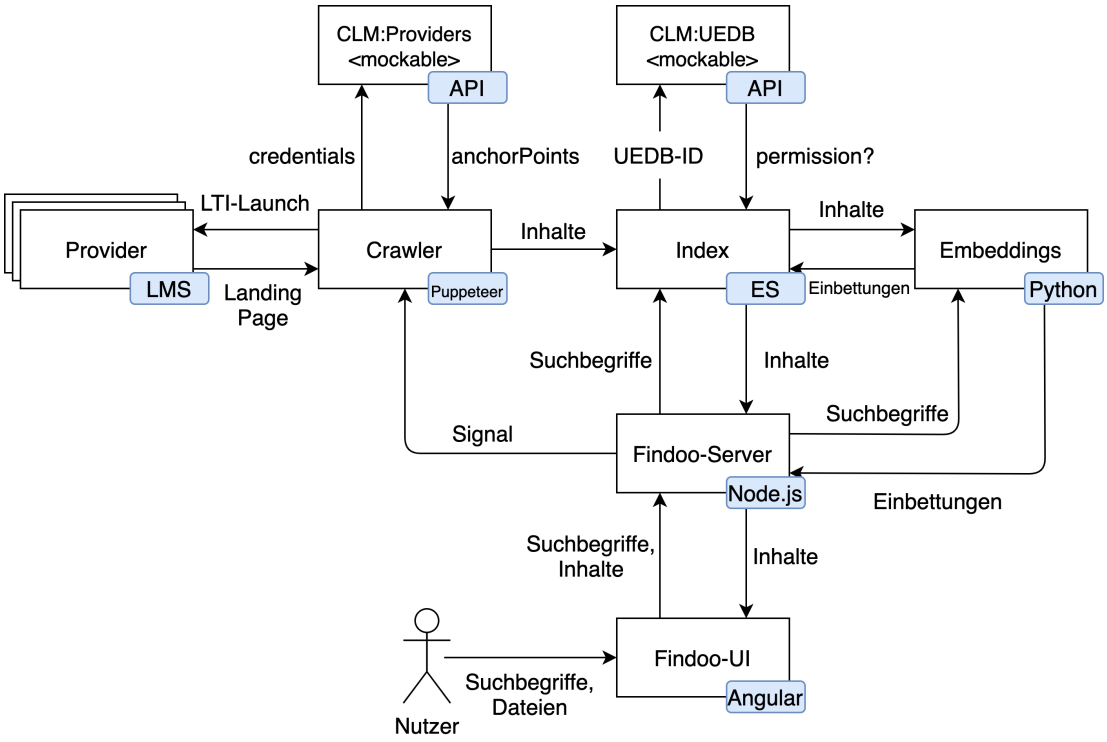


Abbildung 5.1: Die Komponenten und die jeweils verwendeten Technologien in der Übersicht.

dem System nur als Ankerpunkte, bestehend aus einer Tool ID und einer Provider ID, übergeben. Will der Crawler einen Launch ausführen, muss die Anfrage ebenfalls über die CLM erstellt werden. Genauer wird dort der Body erstellt und die URL preisgegeben, nur dann wird anschließend der Body vom Crawler an das LMS weitergeleitet. Außerdem verwaltet die CLM ein Nutzersystem, das die für einen gegebenen Account sichtbaren Tools kontrolliert. Das System ist auch lokal, d.h. ohne CLM lauffähig. In diesem Fall wird die CLM durch einen Mock ersetzt, der die für das Crawling relevante Funktionalität ebenfalls implementiert und eine beliebige Menge von Tools einlesen kann. Hier müssen die Details der LMS aber bekannt sein.

Abbildung 5.2 zeigt die Integration des Systems in die CLM-Plattform Ephesos [IOS21]. Findoo-UI ist dort zunächst als normales Tool eingebunden und wird via LTI-Launches geöffnet. Dies stellt aber zunächst nur eine einseitige Kommunikation sicher. Will der Nutzer ein gefundenes Suchergebnis in Ephesos öffnen, ist das zunächst nicht möglich, denn ein Tool kann, da der Outcomes-Service von LTI nicht unterstützt wird, nicht mit dem Portal kommunizieren. Hier kommt dann eine der Regeln aus Sektion 5.4 zum Einsatz, der *Fallback-Standard*. Das Frontend schickt via des PostMessage-Standards eine Nachricht an das ihn einbettende Portal,

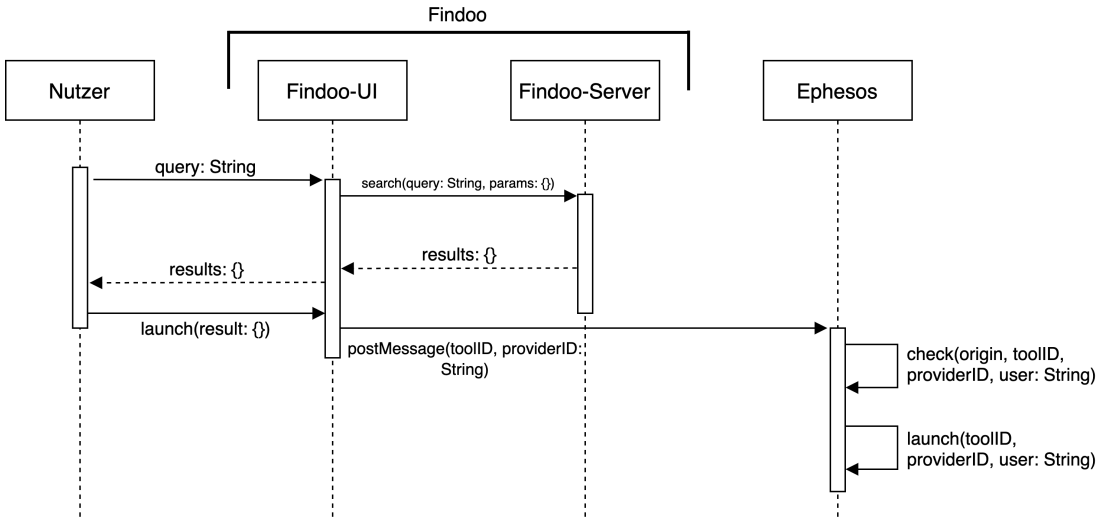


Abbildung 5.2: Die Integration des Systems in Ephesos [IOS21] durch einen LTI-Launch und anschließende Post-Messages.

Ephesos kann nun also sicher, wenn auch nicht ohne eigene Unterstützung Nachrichten empfangen. Findoo-UI muss unterdessen nur das Format der Nachricht unterstützen.

5.2 Technische Details

Diese Sektion beschreibt ausgewählte Aspekte der Implementierung und die hiermit gelösten technischen, aber auch konzeptionellen Probleme.

Die Sichtbarkeit von Inhalten Welche der an die CLM [IOS21] angeschlossenen Tools indiziert werden, wird über den Account kontrolliert mit dem Findoo an die CLM angebunden ist. Dieser beeinflusst welche Ankerpunkte bei der entsprechenden Anfrage zurückgeliefert werden. Bei der Nutzung der Suchfunktionalität muss allerdings beachtet werden, dass der suchende Nutzer in der Regel nur auf eine Untermenge dieser Tools zugreifen darf - diese Sichtbarkeitseinschränkungen müssen also berücksichtigt werden. Findoo hat jedoch (schon aus datenschutzrechtlichen Gründen) keinen direkten Zugriff auf den Nutzer-Account, sondern erhält beim Tool-Launch nur eine transparente ID. Die sichtbaren Tools können dann mit dieser ID bei einem separaten Endpunkt an der CLM abgerufen werden. Sind diese bekannt, wird ein *Filtered Alias* erstellt, also eine Sicht auf den gesamten Index, der nur die sichtbaren Inhalte enthält. Dies hat den Vorteil, dass bei Suchen auf diesem Alias die Sichtbarkeitseinschränkungen stets implizit berücksichtigt werden.

Find-Body-Heuristik Das Indizieren von Webseiten in Suchmaschinen hat die zentrale Herausforderung, dass (in der Regel) nach dem natürlichsprachlichen Body gesucht wird und nicht nach dem eigentlichen HTML. Zur Identifikation dieses Bodys wird die Implementierung des Firefox-Reader-Mode [Fir21] verwendet, der Ablauf ist grob wie folgt:

1. Zuerst werden Elemente entfernt, die unter keinen Umständen Teil des Body sind, z.B. script-Tags und Formulare.
2. Danach werden Paragraph-Tags betrachtet. Diesen werden dann Scores zugewiesen, abhängig von Kriterien wie der Länge des Inhaltes (als Klartext) oder der Zahl von Satzzeichen etc.
3. Nun folgt der Bubble-Up-Schritt: Die Scores der Paragraph-Nodes der untersten Ebene werden im Rahmen ihrer Parent-Hierachie als gewichtete Summe aggregiert - enthalten sowohl Eltern als auch Kindknoten viel Klartext, ist die Summe höher als wenn Text nur in einer Ebene vorkommt.
4. Der Body setzt sich dann aus den am höchsten gewichteten Inhalten zusammen.

Als generische Heuristik ist die Qualität der Ergebnisse von der Struktur der konkret betrachteten Seite abhängig, ist aber für den Anwendungsfall dieser Arbeit sehr gut. Diese wird in Kapitel 6 mithilfe des Jaccard-Index evaluiert. Als Alternative wurde versucht, angelehnt an das robuste Crawling, das Konzept der Visible Text Selectors auch hier anzuwenden. In diesem Framework würde der Nutzer eine Menge von proprietären CSS-Selektor-ähnlichen Marker spezifizieren, die den Bereich des Body auf der Website spezifizieren. Hier wären Ausdrücke mit der Semantik wie „der gesamte Text zwischen dem Impressum und dem Navigationsmenü mit den Worten A,B oder C“ möglich. Der Implementierungsaufwand dieses Systems im Vergleich zum Gewinn an Robustheit ist allerdings unverhältnismäßig, insbesondere im Kontext der Performance der automatischen Heuristik.

Concept Extraction via DBpedia Spotlight Der DBpedia-Spotlight-Algorithmus extrahiert DBpedia-Konzepte aus einem gegebenen Text [Men+11; Dai+13]. Genauer wird der Text auf *surface forms* untersucht - Ankertexte s, die in Wikipedia-Artikeln auf Ressourcen r verlinken (können). Jedes Auftreten eines solcher Ankertexten wird als *evidence of occurence* $o = (r,s)$ formalisiert. Die Stärke dieser Beweise wird abgeschätzt via der Formel: $P(r|s) = n(s,r)/n(s)$. Die Wahrscheinlichkeit, dass das Auftreten von s eine Beziehung zu der Ressource r besagt ist gleich der Anzahl der Vorkommnisse von s, bei denen in Wikipedia diese Assoziation besteht ($n(s,r)$), durch die Gesamtanzahl der Vorkommnisse $n(s)$. Der Spotlight-Algorithmus läuft in 3 Schritten ab [Men+11]:

1. *Spotting*: Auf Basis eines Lexikons und eines String-Matching-Algorithmus werden alle Worte, die potentiell zu einem bekannten Konzept gehören markiert. Die Menge potentieller Matches wird dabei mit einem Hidden-Markov-Model-basierten Part-of-Speech-Tagger verringert, indem irrelevante Satzteile (wie Adverben, Präpositionen etc.) von der Spotting-Analyse ausgenommen werden.
2. *Candidate Selection*: Vor der eigentlichen Konzeptbestimmung werden für alle gefundenen surface forms die jeweils möglichen (aber i.A. unterschiedlich wahrscheinlichen) Konzepte im DBpedia-Datensatz ausgewählt. Dies kann auch parametrisiert ablaufen, sodass Kandidaten mit sehr geringer Wahrscheinlichkeit nachfolgend gar nicht betrachtet werden.
3. *Disambiguation*: Nun muss für jede Surface-Form das wahrscheinlichste Konzept bestimmt werden. Dazu wird das Konzept der *inverse candidate frequency* verwendet, gegeben in Gleichung 5.1 für ein Wort w_j, die Menge der potentiellen Kandidaten R_s für eine surface form s und die Zahl der Ressourcen in R_s, die mit w_j assoziiert sind $n(w_j)$. Nun ist es die Aufgabe des Disambiguation-Schrittes den Kontext um die gewichteten Kandidaten zu betrachten und das wahrscheinlichste Konzept zu bestimmen. Für die Kontext-Ähnlichkeit wird ein Vektor-Modell mit der Cosinus-Ähnlichkeit verwendet.

$$ICF(w_j) = \log \frac{|R_s|}{n(w_j)} = \log |R_s| - \log n(w_j) \tag{5.1}$$

Für Details und Erweiterungen sei auf die Arbeiten von Mendes und Daiber et al. [Men+11; Dai+13] verwiesen.

DBpedia Lookup Der DBpedia-Lookup-Service [DBp21] implementiert eine Keyword-basierte Suche der DBpedia-Ontologie, dabei werden Titel und Abstracts betrachtet. Es ist ebenso möglich Einträge nach ihrer Ontologie-Klasse zu filtern. Als Relevanz-Metrik wird zusätzlich ein PageRank durchgeführt: Einträge werden nach der Zahl ihrer eingehenden Referenzen sortiert. Die Ergebnisse liefern eine Reihe von Feldern, insbesondere wichtig sind im Rahmen dieser Arbeit die assoziierten Konzepte.

Spracherkennung via N-Grams In der Formulierung von Cavnar et al. [CT+94] versteht man unter N-Grams konsekutive Abschnitte der Länge N einer gegebenen Zeichenkette. Eine Zeichenkette der Länge k hat $k + 1$ (ggf. mit Leerzeichen aufgefüllte) N-Grams. Die Menge der häufigsten N-Grams, genau wie die Menge der häufigsten Worte, sind charakteristisch für eine Sprache und können deswegen zur Erkennung derselben verwendet werden. Der Ablauf ist dann wie folgt: Für einen gegebenen Text werden die l häufigsten N-Grams berechnet, das Ergebnis ist das *Dokumentenprofil*. Nun werden diese gezählt und mit einer Reihe von

einmalig vorberechneten Sprachprofilen verglichen. Hier kommt die Out-Of-Place-Metrik zum Ansatz, die für jedes Paar aus Dokumentprofil und Sprachprofil eine Kostenfunktion berechnet. Zuerst werden die N-Grams der beiden Profile nach ihrer Frequenz sortiert. Nun wird für jedes N-Gram im Dokumentenprofil dessen Platzierung im Sprachprofil gesucht. Die Kosten für ein N-Gram sind die Distanz der so zugeordneten Platzierungen, die Gesamtkosten sind die Summe der Kosten aller N-Grams. Am Ende wird die Sprache ausgegeben, deren Profil die geringsten Kosten verursacht [CT+94].

Performance Ein kritischer Aspekt jeder Suchmaschine ist deren Geschwindigkeit, hier wird diese durch eine sorgfältige Implementierung der Aspekte des Information Retrieval-Algorithmus erzielt. Konkret bedeutet das:

- Die Implementierung der Wort-Einbettungen findet vektorisiert statt und nutzt, soweit möglich, PyTorch als performantes Framework.
- Die Ontologie-Komponenten verwenden eine lokale DBpedia-Instanz in einem Docker-Container.
- Die rechenintensiven Suchoperationen finden ausgelagert in Elasticsearch und den dort angebotenen nativen Bindungen statt. Diese sind dort in der Script-Sprache Painless als *Script Score* implementiert. Painless verwendet eine Java-ähnliche Syntax und kompiliert zu Java-Byte-Code. Zudem wird intensiv Caching von Suchanfragen und Filtern verwendet.

Non-Standard Launch Der LTI-Standard spezifiziert als Reaktion auf einen Basic Launch Request, dass das zugehörige Tool geöffnet wird. Welche URL konkret geöffnet wird ist dem Tool Provider überlassen - Tools werden also als atomare Elemente behandelt. Insbesondere im LMS-Kontext ist das aber in der Regel nicht der Fall. Der Crawler identifiziert Seiten bzw. Inhalte, die Teil eines Kurses sind, aus LTI-Sicht ist aber nur der Kurs an sich ein Tool. Das Öffnen der konkreten Suchergebnisse ist mit dem Standard-Launch nicht möglich, denn dort wird stets die Landing Page des Kurses geöffnet. Dieses Problem wird im Rahmen des Editors via *asynchroner Navigation* gelöst. Beim Klick auf ein Suchergebnis wird das Tool zunächst via eines Standard-Launches geöffnet, eine Session wird etabliert und die Standard-Navigation läuft ab. Nach dieser wird nun die eigentliche URL des Inhaltes direkt aufgerufen. Die nötige Authentifizierung ist nun bereits geschehen (und in einem Session-Key o.ä. gespeichert) und der Deep Link kann somit direkt geöffnet werden.

Near Duplicate Detection via TLSH Um Redundanz zu vermeiden, sollten Inhalte stets nur einmal indiziert werden. Im Allgemeinen ist für die Duplikat-Erkennung aber ein einfacher

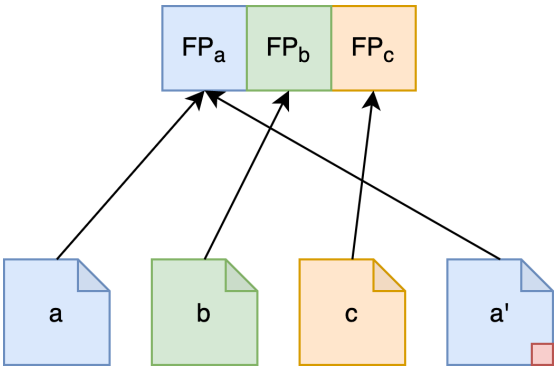


Abbildung 5.3: Near Duplicate-Detection durch die TLSH-Funktion.

URL-Vergleich nicht ausreichend. Da sowohl die Struktur der LMS als auch der Inhalte selbst unbekannt ist, kann es prinzipiell dazu kommen, dass der gleiche Inhalt (beispielsweise via Skript oder URL-Argumente) mehrmals unter unterschiedlichen URLs gefunden wird. Folglich muss ein inhaltsbasierter Vergleich stattfinden. Hierzu werden die Bodies (und nicht der Code der Website) verwendet. Deren heuristische Identifikation ist allerdings nicht deterministisch und außerdem kann es durchaus sein, dass kleine Teile der Website nur manchmal sichtbar sind, den eigentlichen Inhalt aber nicht entscheidend verändern. Diese Fälle müssen alle abgedeckt werden, es kann also nicht auf strenge Identität geprüft werden, sondern auch auf *near-duplicates*. Zusätzlich ist es nicht optimal alle Inhalte stets im Speicher zu halten.

Der hier entworfene Lösungsansatz, dargestellt in Abbildung 5.3, berechnet via einer Hash-Funktion für jeden indizierten Body einen kleinen Fingerprint fester Länge und verwendet diesen als Vergleichsbasis. Die zentrale Schwierigkeit ist die Wahl und Verwendung einer anti-kryptographischen-Hashfunktion - einer Funktion, bei der ähnlicher Input auf ähnlichen Output abgebildet wird. Sind die Werte zweier Bodies ähnlich, handelt es sich um *near duplicates*. Hier kommt die von Oliver et al. [OCC13] entworfene Version einer Timed Locality Sensitive Hash-Funktion zum Einsatz. Diese läuft grob ab wie folgt:

- Der Input wird als Byte-String betrachtet, die Bits werden mit einem Sliding Window der Größe 5 einem von 128 Buckets zugeordnet.
- Danach werden die Quartilpunkte q_1,q_2,q_3 berechnet, sodass jeweils 75,50 und 25 Prozent aller Buckets mindestens q_i Elemente enthalten.
- Nun wird ein aus 256 Bit bestehender Hash berechnet, indem jedem Bucket zwei Bits zugeordnet werden, abhängig von dessen Quartil, und anschließend die Bits konkateniert werden.

5.3 Modellauswahl

Bei der Verwendung der semantischen Einbettungen gibt es zwei zentrale Freiheitsgrade: Der erste ist die Wahl des konkreten, vortrainierten Modells. Dieser Schritt ist essenziell, denn wie an anderer Stelle bereits betont ist kein Training des Modells möglich, da die Anwendungsdomäne unbekannt bzw. nicht fest ist: die *General Purpose*-Eigenschaft. Das Pre-Training des Modells ist also die wesentliche Performance-Grundlage. Der zweite betrifft die Aggregationsmethode. Die getroffenen Entscheidungen werden nachfolgend motiviert, die eigentliche Performance des Systems wird in Kapitel 6 evaluiert.

Die Aggregationsmethode Das Konzept der Einbettung eines Dokumentes ist zunächst nicht wohldefiniert. Ein Dokument kann entweder als ganze Einheit betrachtet werden (*document level*), als eine Menge von Sätzen (*sentence level*) oder als eine Menge von Tokens bzw. Wörtern (*token level*). Desweiteren ist der konkrete Vergleich mit einem Query-Satz auch ein Freiheitsgrad. Mit diesen Konzepten können nach Ghosh [Gho20] die folgenden konkreten Einbettungs- bzw. Vergleichsmethoden definiert werden:

- *Document Level*: Bette sowohl das Dokument als auch die Query als einen Satz ein und vergleiche.
- *Document Sentence Level*: Bette die einzelnen Sätze des Dokuments ein und bilde anschließend den Durchschnitt, vergleiche diesen mit der Query.
- *Average Sentence Level*: Bette die einzelnen Sätze des Dokuments ein, vergleiche diese isoliert mit der Query und bilde anschließend den Durchschnitt.
- *Max Sentence Level*: Bette die einzelnen Sätze des Dokuments ein, vergleiche diese isoliert mit der Query und finde anschließend das Maximum.
- *Max Token Level*: Bette die einzelnen Tokens des Dokuments ein, vergleiche diese mit allen Tokens der Query, anschließend wird das Maximum dieser Vergleiche gebildet.
- *Average Token Level*: Bette die einzelnen Tokens des Dokuments ein, vergleiche diese mit allen Tokens der Query, anschließend wird der Durchschnitt dieser Vergleiche gebildet.
- *Median Token Level*: Bette die einzelnen Tokens des Dokuments ein, vergleiche diese mit allen Tokens der Query, anschließend wird der Median dieser Vergleiche gebildet.
- *Max Average Token Level*: Bette die einzelnen Tokens des Dokuments ein, vergleiche mit einem Token der Query und bilde den Durchschnitt. Dies wird für alle Tokens der Query wiederholt und dann das Maximum gebildet.
- *Average Max Token Level*: Bette die einzelnen Tokens des Dokuments ein, vergleiche mit einem Token der Query und bilde das Maximum. Dies wird für alle Tokens der Query wiederholt und dann der Durchschnitt gebildet.

Methode	pos. Cosinus-Sim.	neg. Cosinus-Sim.	Δ
Max Average Token Level	0.796	0.578	0.218
Document Sentence Level	0.282	0.13	0.152
Max Sentence Level	0.356	0.218	0.138
Max Token Level	0.863	0.754	0.109
Document Level	0.144	0.041	0.103
Median Sentence Level	0.136	0.07	0.066
Average Sentence Level	0.132	0.075	0.057
Median Token Level	0.263	0.255	0.008
Average Token Level	0.353	0.347	0.006
Average Max Token Level	0.505	0.509	-0.004

Tabelle 5.1: Die zur Auswahl stehenden Aggregationsmethoden und deren Performance im durchgeführten Experiment.

Die Experimente zur Methoden-Auswahl werden auf dem in Kapitel 6 erklärten E-Learning-Datensatz ausgeführt und laufen ab wie folgt: Die Dokumente werden mit den verschiedenen Methoden eingebettet, für jede Kategorie werden die Dokumente dann via der Cosinus-Ähnlichkeitsmetrik mit Beispiels-Anfragen verglichen, die eine gesamte Kategorie beschreiben. Konkret wurden für die Anfragen einfach die Namen der Kategorien verwendet. Es sind also alle Dokumente in der Kategorie relevant und alle Dokumente außerhalb der Kategorie irrelevant. Das Ziel sind dabei folglich hohe Ähnlichkeitswerte des Dokuments mit der eigenen Kategorie (positive Cosinus-Similarity) und niedrige mit anderen (bezeichnet als negative Cosinus-Similarity). Die Qualität einer Methode kann nun durch das Delta (Δ) der positiven und der negativen Cosinus-Similarity beschrieben werden, je größer desto besser. Wie in der Tabelle 5.1 dargestellt, hat die *Maximum Average Token*-Methode das größte Delta. Dieses wird allerdings nicht im System verwendet, da für deren Implementierung die Einbettung jedes einzelnen Tokens jedes Dokumentes gespeichert werden müsste. Die Wahl fällt also auf die zweitbeste Methode: *Document Sentence Level*.

Das zugrundeliegende Modell Aus theoretischer Sicht können (fast) alle NLP-Modelle entweder direkt oder mithilfe naheliegender Modifikationen für die Berechnung von Einbettungen benutzt werden. Wie von Reimers et al. [RG19] beschrieben, ist allerdings die Performance solcher modifizierten Modelle im neuen Kontext der Suche in der Regel schlechter als die von älteren, aber für Suchaufgaben konzipierte Modellen. So ist zum Beispiel das derzeit beste Modell für NLP-Aufgaben, BERT [Dev+19], weniger gut für Suchen geeignet als GloVe [PSM14]. Für die Anwendung in semantischen Suchen wurde von den Autoren Sentence-BERT [RG19] ent-

worfen. Insbesondere mit Hinblick auf die spätere Verwendung des Modells, der Approximation der Semantik von einzelnen Stichworten und (gleichzeitig) von längeren, natürlichsprachigen Dokumenten liegt die Verwendung von Sentence-BERT nahe. Die entsprechende Architektur bzw. Trainigsmethode wurde bereits in Kapitel 3 diskutiert und ist speziell für den Vergleich der Semantik von Sätzen ausgelegt. Hier stehen im Rahmen der zugehörigen Python-Bibliothek[1] aber wiederum mehrere Modelle zur Auswahl, für den konkreten Anwendungszweck sind das die in Tabelle 5.2 dargestellten. Die Namen geben jeweils einen Hinweis auf die verwendeten Trainigsdaten bzw. -Methoden, werden an dieser Stelle aber nicht weiter thematisiert. Das Experiment zur Modellauswahl findet analog zur Methodenwahl statt. Die Dokumente werden mit der oben gewählten Methode des Document Sentence Level mithilfe verschiedener Modelle eingebettet und mit relevanten bzw. irrelevanten Anfragen verglichen. In der Tabelle sind die Durchschnittswerte der positiven und negativen Cosinus-Ähnlichkeit über alle Dokumente dargestellt. Die besten Modelle haben das größte Delta zwischen diesen beiden Metriken. Dabei handelt es sich um STSB XLM-R Multilingual, STSB Distilbert und STSB BERT. Als zweiter Test wurde ein Teil der Evaluation aus Kapitel 6 durchgeführt - das Berechnen der Mean Average Precision über 30 Anfragen. Aus der Kombination dieser MAP und dem Delta kann nun das beste Modell gewählt werden, dabei handelt es sich um *STSB Distilbert*. Weiter gilt es folgendes zu beachten: Die Deltas sind recht gering, das ist bedingt durch die in Kapitel 4 dargestellte Eigenschaft, dass sich die Ähnlichkeit von semantischen Einbettungen fast immer im Intervall $(-0.1, 0.8)$ bewegt, also nicht den vollständigen Wertebereich ausnutzt. Das beste Modell ist STSB Distilbert, was etwas überraschend ist, da es sich dabei um eine auf Performance ausgelegte Architektur handelt. Zudem ist die Performance von STSB XLM-R Multilingual recht gut, insbesondere da es sich dabei um um ein Modell handelt, das mit mehreren Sprachen umgehen kann.

5.4 Lessons Learned: Robustheit und Interoperabilität

Diese Sektion fasst Regeln und Best Practices für die Entwicklung und Verwendung von interoperablen Systemen zusammen, die im Rahmen der Arbeit festgestellt, formuliert und angewendet wurden.

Minimale Annahmen In interoperablen Systemen sind eine Reihe von Eigenschaften gegeben, die in internen Informationssystemen einer Organisation teilweise nur eine untergeordnete Rolle spielen. Insbesondere die Heterogenität von Nutzern und Entwicklungsteams kann zur

1 https://www.sbert.net

Modell	pos. Cosinus-Sim.	neg. Cosinus-Sim.	Δ	MAP
STSB Distilbert	0.326	0.113	0.213	0.489
STSB BERT	0.336	0.133	0.203	0.45
STSB XLM-R Multilingual	0.374	0.14	0.234	0.407
Cross-EN-DE Roberta	0.222	0.076	0.146	0.314
STSB Roberta	0.356	0.189	0.167	0.306
Distiluse Multilingual Cased	0.24	0.053	0.177	0.218
BERT NLI Mean Tokens	0.525	0.355	0.17	0.191
Distilbert NLI Mean Tokens	0.532	0.383	0.149	0.175

Tabelle 5.2: Die zur Auswahl stehenden Einbettungs-Modelle und deren Performance im durchgeführten Experiment.

Herausforderung werden. Es sollten keine Annahmen über die Interna eines Moduls oder gekoppelter Systeme getroffen werden - die Performance der unterliegenden Datenhaltung, welche Bedeutung extern sichtbare Variablen wie IDs intern haben, welche Datenhaltung dem System unterliegt etc. Zudem sind die verwendeten Technologien zwar (notwendigerweise) kompatibel, aber die Intention der getroffenen Entscheidungen wird in der Regel nicht eingehalten. Typische Symptome treten in der Form von Anti-Patterns auf. Dazu zählen:

- *Refused Bequest* [Fow18]: Ein Parameter oder eine Funktion kann nicht sinnvoll implementiert werden. LTI-Beispiele sind konstante, irrelevante Nutzer oder nicht spezifizierte Return-URLs.
- Verletzungen des *Open-Closed-Principle* sowie des *Dependency Inversion-Principle* nach [Mar02]: Parameter werden missbraucht (interne Ids, die extern gebraucht werden) oder es findet Nicht-Standard-Kommunikation statt. Die Post-Messages, die vom Empfänger von Nachrichten zum Sender dieser Nachrichten zurückgesendet werden (müssen) sind ein Beispiel im CLM- und Ephesos-Kontext.
- Verletzungen von *Don't Repeat Yourself*: Funktionen müssen erneut bzw. redundant implementiert werden, weil deren Nutzungsszenarien nicht komplett oder exakt genug spezifiziert wurden.

Außerdem gilt generell das Problem von divergenten Evolutionsszenarien. Dieses kann durch das Tolerant Reader-Pattern oder gar Consumer-driven Contracts gelöst werden [Dai12]. Gemäß diesen werden unbekannte Parameter ignoriert oder die Funktionalität erst zur Nutzungszeit ausgehandelt. Dies sichert zwar die Evolutionsfähigkeit, kann aber auch zu transienten Fehlern führen, z.B. erfordert ein LTI-Launch bei Moodle weniger Parameter als ein Launch bei Open edX, diese semi-optionalen Parameter können die Fehlersuche erheblich erschweren.

Microservices Der Architekturstil der Microservices ist für interoperable Systeme besonders geeignet. Als Microservice bezeichnet man eine Software-Komponente, die als eigener Prozess läuft und mit dem Rest der Applikation via einer API kommuniziert. In der Regel implementiert ein solcher Service eine kleine, kohäsive Menge von Funktionalitäten. Gemäß Fowler [Fow14] sind entscheidende Eigenschaften von Microservices:

- Unabhängiges Deployment
- Unabhängige Skalierbarkeit
- Heterogene Verwendung von Technologien
- Verwaltung durch verschiedene Teams

Diese Eigenschaften sind offensichtlich geeignet für den Kontext der Interoperabilität mit der inhärenten Heterogenität von Technologien, Konventionen und Ökosystemen.

On-Demand Mocking Ein weiteres Phänomen interoperabler Systeme ist die kombinatorische Explosion von Nutzungskontexten. Ein Modul bzw. Service muss mit mehreren anderen Modulen kommunizieren und deren Anforderungen erfüllen. Dies wird zum Problem, wenn diese Anforderungen zueinander inkompatibel oder redundant sind. Im konkreten Fall dieser Arbeit gibt es zwei Anwendungsfälle des Systems: Einen lokalen Modus ohne Common Learning Middleware (CLM) aber mit Editor-Möglichkeiten und einen Online-Modus, in dem das System nur als Suchmaschine als Teil von CLM fungiert. Naiv müsste hier Funktionalität redundant implementiert werden und verschiedene Produktlinien entstehen. Eine bessere Lösung ist das *On-Demand-Mocking*. Der lokale Modus ahmt hierbei das externe Interface des Online-Modus nach, es wird also ein Mock der CLM verwendet. Diese Komponente ist nun immer Teil des Systems, wird aber nur bei Bedarf auch verwendet. Der Grundsatz lautet, dass nur eine Komponente ausgetauscht werden muss, nicht mehrere (oder gar alle). Die Konfiguration kann dann einfach über Umgebungsvariablen erfolgen.

Location, Location, Location Im Rahmen des Designs von Systemen, die via einer öffentlichen API mit (externen) Klienten kommunizieren, ist es im Allgemeinen nicht klar wo gewünschte oder nötige Funktionalität implementiert werden soll. Aus Sicht des Klienten ist es zunächst zwar von Vorteil, wenn dieser nur wenige Parameter an einen Endpunkt der API senden muss, allerdings kann dies die Nutzbarkeit des Systems auch einschränken. Wenn zum Beispiel der Nutzer einen Basic-Launch des Standards Learning Tools Interoperability (LTI) [IMS19a] selbst schreibt und nur via der API an das LMS weitergibt, kann dieser genau die gewünschten Parameter hinzufügen. Die API hat von Änderungen, Besonderheiten oder Erweiterungen der Parameter keine Kenntnis und muss nicht angepasst werden. Wenn die API

die Launches selbst konstruiert, hat der Benutzer aber im Erfolgsfall sehr viel weniger Aufwand. Ausschlaggebend für die Verteilung der Funktionalität ist also wie oft Änderungen nötig sind. Der Aufwand auf Nutzerseite hängt außerdem von dessen verwendeten Technologien ab. Beispielsweise ist die Verwendung der OAUTH1-Signatur in Python trivial via einer Bibliothek möglich, in JavaScript aber nicht.

No one metric to rule them all Heterogene Systeme implizieren in der Regel auch heterogene Nutzer. Deshalb sind Qualitätsmetriken, die für eine Komponente sinnvoll scheinen nicht immer systemweit anwendbar. Für das entworfene Information Retrieval-System bedeutet das das folgende: Der Suchmodus dient der Content Discovery - der Nutzer möchte Inhalte finden, die sich konkret auf ein bestimmtes Thema beziehen. Dies lässt sich durch eine hohe Präzision ausdrücken. Im Editor-Modus wird die Suche in der Regel von einem Domänen-Experten ausgeführt - dieser wird eine größere Kapazität haben zwischen relevanten und irrelevanten Inhalten zu unterscheiden und will ggf. auch Inhalte betrachten, die weniger streng relevant sind, z.B. als ergänzende Information. Hier ist also eher der Recall wichtig. Die verschiedenen Anforderungen lassen sich durch anpassbare Optionen und Parameter beantworten. Insbesondere lässt sich die semantische Suche anpassen oder deaktivieren. Als genereller Grundsatz ist aber festzuhalten: Die Architektur soll sich den Use Cases anpassen, nicht andersherum.

Fallback-Standards Oft werden Standards nicht komplett verwendet oder in einem anderen Kontext als ihr Design vorsieht. Dies kann zur Folge haben, dass sie nicht für alle Anforderungen bzw. Funktionalitäten verwendet werden können - so kann LTI in einem System z.B. den Launch eines Tools übernehmen, aber nicht die anschließende Kommunikation, wenn dieser Teil des Standards (der *Outcome Service*) nicht implementiert ist. Nun ist eine proprietäre Lösung zu finden. Hier gibt es aber auch Grundsätze und Regeln zu beachten, insbesondere können und sollten auch in proprietären Lösungen andere, ergänzende Standards verwendet werden. Im konkreten Anwendungsfall kommuniziert beispielsweise die Suchmaschine nach einem standardkonformen LTI-Launch via Post-Messages mit ihrer Umgebung - dies stellt die Sicherheit der Kommunikation sicher und macht diese flexibler, wenn auch nicht komplett plattformagnostisch.

Agile Prozesse: Kommunikation und Flexibilität In interoperablen Systemen sind flexible Prozessstrukturen besonders wichtig. Es liegt in der Natur der Sache, dass hier Entwicklungsteams mit verschiedenen Ansichten, Interessen, Fähigkeiten oder Zeitplänen kooperieren müssen. Risiken müssen also früh entdeckt und mitigiert werden und die Definitionen von Problemen und Zielen kann nicht in Isolation entstehen. Ein zentrales Risiko ist das der Blocka-

den. Diese treten auf, wenn ein Modul vollkommen abhängig ist von Funktionalität, die noch nicht existiert. Entweder verzögert sich nun das ganze Projekt oder das verzögerte Team muss Annahmen über das zukünftige Modul treffen, die sehr unsicher sind und entsprechend redundante Mock-Schnittstellen entwickeln, die am Ende keinerlei Mehrwert liefern. Eine wichtige Ausnahme hierbei ist das oben erwähnte *On-Demand-Mocking*. Ein solches Mock-Objekt kann recht einfach und flexibel modifiziert werden und wird auch nach Projektende verwendet.

Standardize the Standard Das Konzept eines Standards ist in der Regel überladen: Es gibt fast immer verschiedene Versionen. Diese können sich nur gering oder komplett unterscheiden, diese können zueinander kompatibel sein oder nicht. Ein Beispiel für Kompatibilität ist das Common Cartridge-Format im Vergleich zum Thin-Cartridge-Format [IMS15]. Manche Versionen sind nur für Spezialfälle geeignet, manche sind echte Weiterentwicklungen oder beinhalten nur syntaktische Veränderungen. Oft haben sich auch nur manche Versionen durchgesetzt, was massive Implikationen auf den Nutzen dieser hat. Die Mehrdeutigkeit von Standards erschwert die Entwicklung entsprechend. Ein weiterer zu nennender Aspekt ist das Konzept des *spirit of the law* gegenüber dem *letter of the law*. Oft implementieren Systeme Standards nur soweit bis die Zertifizierung erfolgen kann - ggf. werden so ganze Teile der Spezifikation nie anwendbar sein. Im konkreten Anwendungsfall ist das LTI-Konzept des *content item requests* [IMS19a] ein Beispiel - dies würde eine strenge Trennung der LMS-Struktur und der Lerninhalte ermöglichen wird aber von (fast) keinem System unterstützt. Hier muss entweder manuell via Plugins nachgerüstet oder auf andere Technologien ausgewichen werden.

A-Priori Zieldefinition Jedem interoperablen System unterliegt ein stetiges Evolutionsbedürfnis, da die (potenziellen) Quellen von Anforderungsänderungen mit der Zahl der Subsysteme ansteigen. Dies kann dazu führen, dass das *YAGNI*-Design-Prinzip [Mar02] verletzt wird und eine Komponente entsteht, die vor allem auf mögliche Anforderungen und nicht ausreichend auf tatsächliche Anforderungen ausgelegt ist. Deswegen muss stets eine klare, unmittelbare Zieldefinition existieren und diese auch kommuniziert werden.

Co-Evolution Die verschiedenen Komponenten eines interoperablen Systems verändern sich in der Regel nicht unabhängig, sondern (ggf. zeitversetzt) zusammen. Dies stellt die Kompatibilität sicher, ohne aufwändige Adapter-Module verwenden zu müssen. Konkret tritt das Phänomen des *Mirroring* auf - ändert sich der Standard verändern, sich die LMS-Implementierungen und somit auch die CLM. Generell sind Änderungen, die nur die CLM betreffen auch sonst an generelle Änderungen gekoppelt - diese werden zum Anlass genommen die Änderungen einzuführen.

Macro-Services Die Architektur eines Softwaresystems kann als die Menge aller getroffener und explizit *nicht* getroffener (d.h. verschobener) Design-Entscheidungen definiert werden [Reu+16]. Generelle Best Practices des Software-Engineering gelten auch im interoperablen Kontext: Anforderungen ändern sich und das Kennzeichen einer guten Architektur ist die Proportionalität zwischen der Größe der Anforderungsänderung und dem Implementierungsaufwand der nötigen technischen Änderungen. Allerdings bleibt diese Proportionalität nicht für jede Architektur konstant, obwohl die ursprüngliche Umsetzung durchaus evolutionsfähig war. Ein Beispiel für dieses Phänomen sind degenerierte Micro-Services: Macro-Services. Dabei handelt es sich um System-Features, die ursprünglich als leichtgewichtige, funktionell limitierte Micro-Services betrieben wurden und über die Zeit immer mehr Funktionalität übernommen haben bis die eigentlichen Vorteile von kleinen Komponenten verloren gegangen sind, obwohl jede einzelne Erweiterung sinnvoll war. Grund für diese Entwicklung sind in der Regel andere Micro-Services, die neue Anforderungen stellen, die z.B. ein starkes Wachstum der Komplexität der Rechteverwaltung zur Folge haben.

Do one thing well Wie bei der Rolle der *Macro-Services* angesprochen sollten einzelne Module eines interoperablen Systems nur eine einzelne Funktion erfüllen. Diese kann intern durchaus komplex sein, nach außen sollte das aber versteckt werden. Oft kann dies dazu führen, dass die Komplexität eines Services antiproportional zu dessen Wichtigkeit im System ist. Hier ist beispielsweise die Service-Beschreibung der CLM sehr kurz und kompakt [IOS21], deren Rolle ist aber trotzdem essenziell. Sie übernimmt die zentrale Konfiguration der angeschlossenen Systeme, stellt rechtliche Compliance sicher und verhindert das *It works on my machine*-Antipattern. Zum Beispiel werden alle LTI-Launchanfragen dort zusammengestellt, es gibt also recht wenige Freiheitsgrade und Fehlerquellen für die (heterogenen) Nutzer. Der zentrale Nachteil dieses Ansatzes ist allerdings der *Single Point of Failure* - funktioniert die Anfragekonstruktion dort nicht, gibt es keine Alternativen.

The spec is never frozen Wie an anderen Stellen bereits betont ist die kontinuierliche Evolution der beteiligten Komponenten eine entscheidende Charakteristik von interoperablen Systemen. Dies hat zur Folge, dass Entwickler einer Komponente nie davon ausgehen können, dass bestimmte Funktionalität in Zukunft nicht von ihrer Seite erforderlich oder von anderer Seite angeboten werden wird. Die Entscheidung ob eine eigene, schnelle, aber ggf. nicht skalierbare Lösung dem Projekt helfen oder Fortschritt verhindern wird ist stets auf Basis von unvollständigen Informationen zu treffen. Um Risiken zu minimieren, sind also die wenigen Feedback-Loops, die beispielsweise durch agile Prozesse und Projektmodelle existieren von essenzieller Bedeutung.

Dominanz von Glue Code Da sich interoperable Systeme insbesondere durch Kommunikation mit anderen Komponenten auszeichnen, ist das Design lokaler Modelle (wie Klassen, Daten-Objekte etc.) weniger wichtig. Die Nutzung von Standards stellt die Interoperabiltität sicher, systemlokale Aspekte werden dadurch beeinflusst. Dies kann zur Folge haben, dass die interne Struktur durch Adapter-Klassen und sogenannten *Glue Code* dominiert wird. Empfangene Daten stammen in der Regel aus einer Industriestandard-Bibliothek, werden lokal (ggf. nur minimal) verändert und dann an einen weiteren Standard weitergegeben.

5.5 Die Nutzerinteraktion

In dieser Sektion wird die Interaktion von Nutzern mit dem Assistenzsystem anhand von Anwendungsfällen erläutert. Die Interaktion findet in einem von zwei Modi statt. Entweder möchte ein Lehrender Inhalte in Kurse einbinden, dies geschieht im Editor, dargestellt in Abbildung 5.5, oder ein Lernender möchte die Suche konventionell benutzen, um Inhalte zur Bearbeitung zu finden. Dies geschieht im reinen Suchmodus, dargestellt in Abbildung 5.9. Beide Teile der Applikation sind entweder über die CLM [IOS21] oder standalone nutzbar.

Anwendungsfall 1: Entdecken neuer Inhalte

1. Ein Lernender möchte die für ihn in der CLM verfügbaren Inhalte durchsuchen ohne daraus einen Kurs zu erstellen, nur um sie zu bearbeiten.
2. Er besucht das Ephesos-Portal und authentifiziert sich.
3. Hier startet er die Suchmaschine über das Portal, dazu wird ein LTI-Launch-Request [IMS19a] an Findoo-UI gesendet. Dieser enthält eine ID des Nutzers.
4. Dieses wird an Findoo-Server weitergeleitet, hier können damit an der CLM die Tools abgerufen werden, auf die der Nutzer Zugriff hat. Nur diese werden in den Suchergebnissen angezeigt.
5. Er tippt einen Suchbegriff in das nun sichtbare Suchfeld ein. Das Suchfenster im Ephesos-Portal ist in Abbildung 5.4 dargestellt.
    a) Während der Eingabe werden ihm passende Vervollständigungen vorgeschlagen.
    b) Der Nutzer filtert die Suchergebnisse nun nach Sprache und Typ.
    c) Nun bestätigt er die Suche per Enter oder durch Klick auf das Lupen-Icon, es werden passende Inhalte angezeigt.
    d) Der Nutzer wählt ein Ergebnis aus, per Klick kann dieses in Ephesos geöffnet werden.

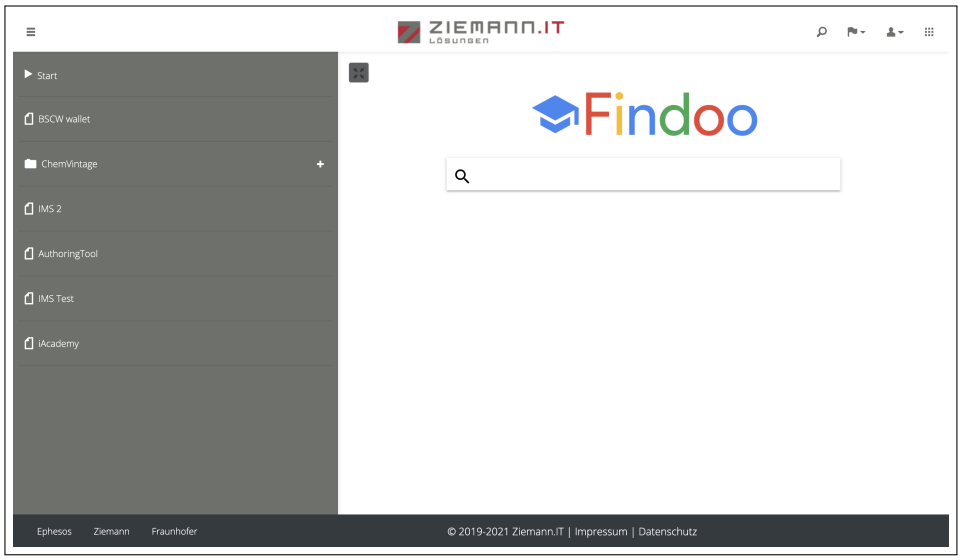


Abbildung 5.4: Das Suchfenster von Findoo als LTI-Tool im Ephesos-Portal.

Anwendungsfall 2: Editieren eines bestehenden Kurses

1. Ein Lehrer möchte einen existierenden CLM-Kurs editieren und um neue Inhalte ergänzen oder bestehende Inhalte verändern. Der Kurs liegt ihm in Form einer Common Cartridge XML-Datei vor.
2. Er hat ein Konzept, wie er den Kurs strukturieren möchte und welche Inhalte enthalten sein sollen.
3. Seine eigenen Inhalte liegen in seinem eigenen LMS (z.B. ILIAS), er möchte aber auch die mittels LTI Deep Linking verfügbaren Inhalte von anderen, thematisch passenden Lernkursen in anderen LMS (z.B. Moodle, Open edX) als Unterseiten einbinden.
4. Er öffnet die Web-Applikation Findoo-UI und lädt die Common Cartridge-Datei hoch.
5. Findoo-UI parst die Datei.
6. Die Datei und die darin enthaltene Kursgliederung können im Rahmen eines integrierten Texteditors oder eines grafischen Baum-Editor angezeigt werden. Diese sind in Abbildung 5.5 dargestellt.
7. Mit Stichworten sucht er nach passenden Inhalten und bekommt nicht nur die eigenen Lerninhalte angeboten, sondern auch passende Inhalte aus anderen LMS.
8. Er fügt diese in die Gliederung in der Baumansicht ein, das Ergebnis ist in Abbildung 5.6 dargestellt.
9. Am Ende kann er die modifizierte Datei herunterladen, als Eingabe für die CLM.

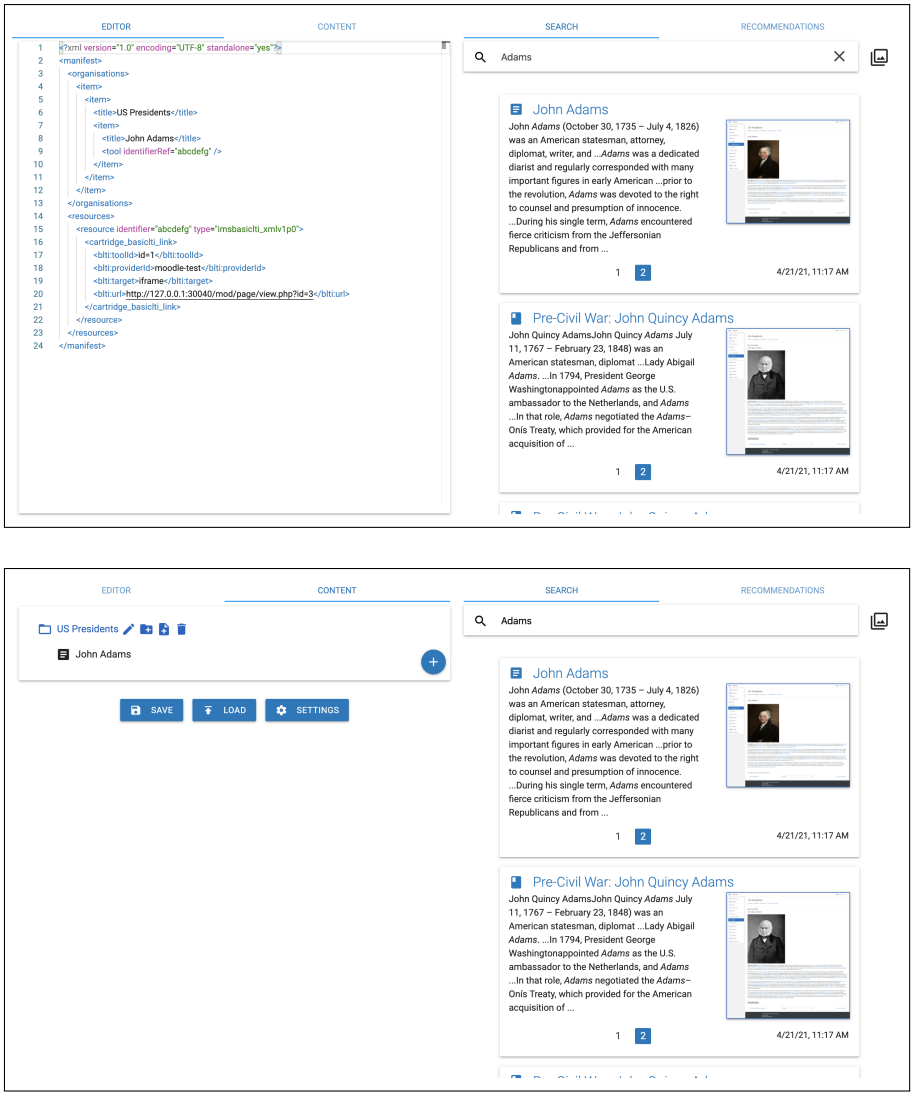


Abbildung 5.5: Die Darstellung von Kursen im Editor: Textbasiert oder als Baumstruktur.

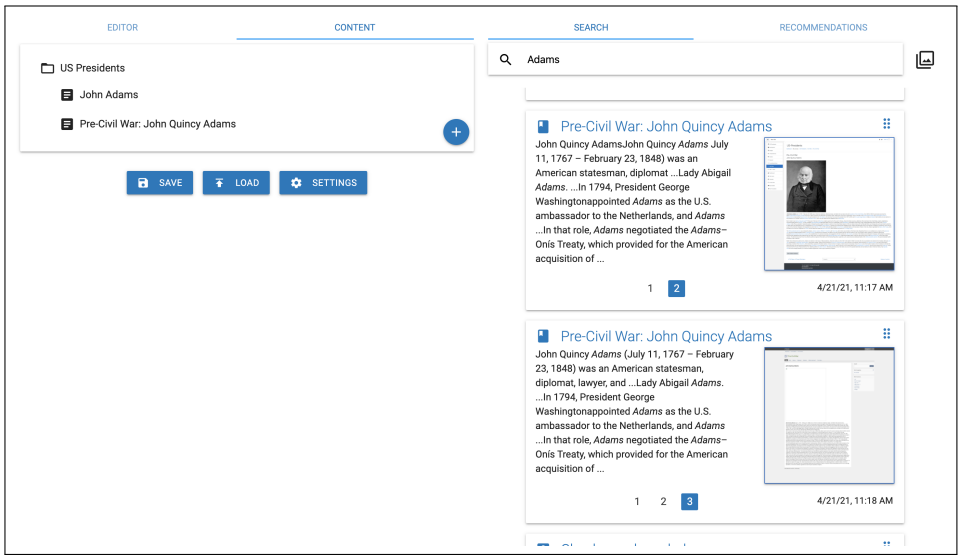


Abbildung 5.6: Das Hinzufügen von Suchergebnissen.

Anwendungsfall 3: Erstellen eines neuen Kurses

1. Ein Lehrer möchte aus existierenden Lernmaterialien einen neuen Kurs erstellen.
2. Der Lehrer öffnet die Web-Applikation Findoo-UI.
3. Ohne Hochladen einer Datei beginnt der Editor mit einer Datei ohne Inhalte, die aber dem Common Cartridge-Format entspricht.
4. Mit Stichworten sucht er nach passenden Inhalten und bekommt nicht nur die eigenen Lerninhalte angeboten, sondern auch die von anderen LMS. Er kann sich sicher sein, dass alle Ressourcen aktuell und verfügbar sind, weil das System kontinuierlich die an die CLM angeschlossenen Systeme überwacht und regelmäßig durchsucht. Für jedes Suchergebnis wird dessen Alter in Form des Indizierungszeitpunktes angezeigt. Inhalte, die älter als 2 Wochen sind werden automatisch gelöscht.
5. Er ist mit den angezeigten Inhalten nicht zufrieden und möchte mehr Inhalte angezeigt bekommen, deswegen aktiviert er die semantische Suche in den Einstellungen, dargestellt in Abbildung 5.7.
6. Nun werden auch Inhalte angezeigt, die thematisch relevant sind, aber die gewählten Suchterme nicht enthalten.
7. Im Recommendations-Tab werden stets proaktive, inhaltlich passende Vorschläge angezeigt, diese verhalten sich wie Suchergebnisse und können entsprechend geöffnet und dem Kurs hinzugefügt werden. Dies ist in Abbildung 5.8 dargestellt.

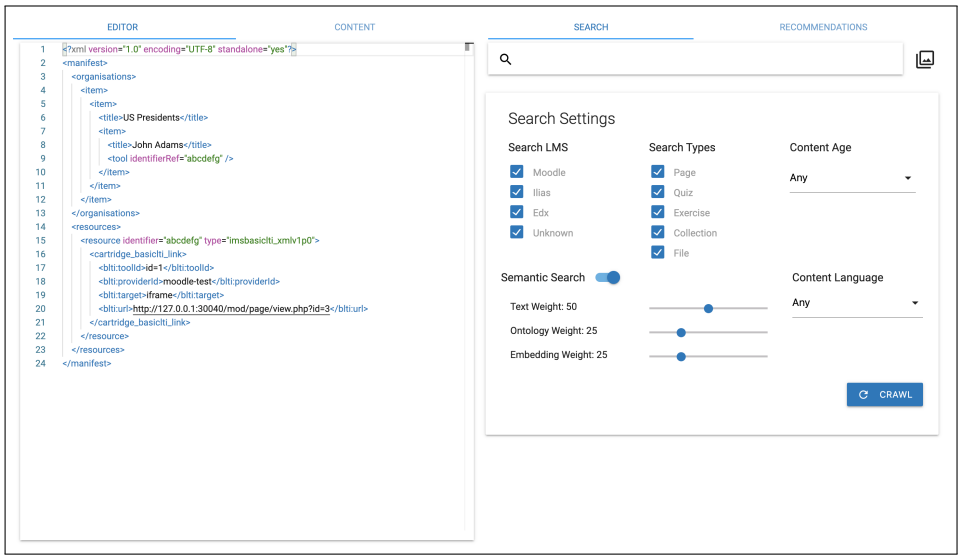


Abbildung 5.7: Das Menü für die Such-Einstellungen.

8. Er fügt diese in die Gliederung in der Baumansicht ein.
9. Am Ende kann er die modifizierte Datei herunterladen, als Eingabe für die CLM.

Anwendungsfall 4: Integration eines neuen LMS

1. Ein Administrator einer Organisation möchte ein neu aufgesetztes oder bereits existierendes LMS in die CLM integrieren und dessen Inhalte in der Findoo-Suche sichtbar machen. Die hierzu nötige Funktionalität existiert in der CLM-API.
   a) Der Administrator gibt eine Liste von allen zur Verfügung stehenden Tools und deren zur Authentifizierung nötigen Credentials an.
   b) Dem Findoo-CLM-Account werden die neuen Tools im User Enrollment Service zugeordnet.
   c) Das Findoo-Server fragt nun beim nächsten Crawling-Vorgang einen (anderen) Endpunkt der Middleware an, dieser gibt eine Liste aller freigegebener Tools als Ankerpunkte zur Verfügung.
2. Ohne weitere Aktionen des Administrators werden die Inhalte beim nächsten Crawling-Vorgang dem Such-Index hinzugefügt und können in der Suche gefunden werden, dargestellt in Abbildung 5.9.

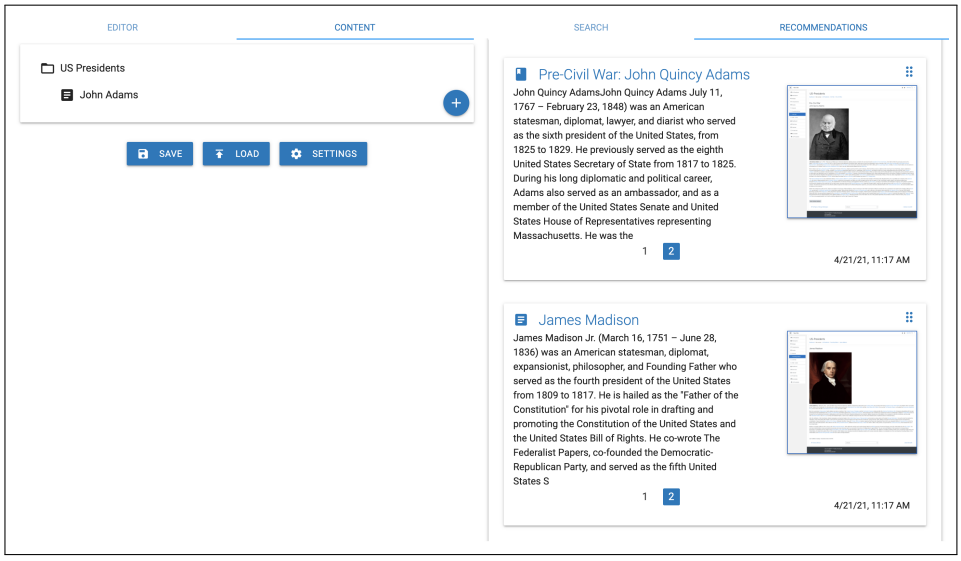


Abbildung 5.8: Die Darstellung inhaltlicher Vorschläge, basierend auf dem aktuellen Kursinhalt.

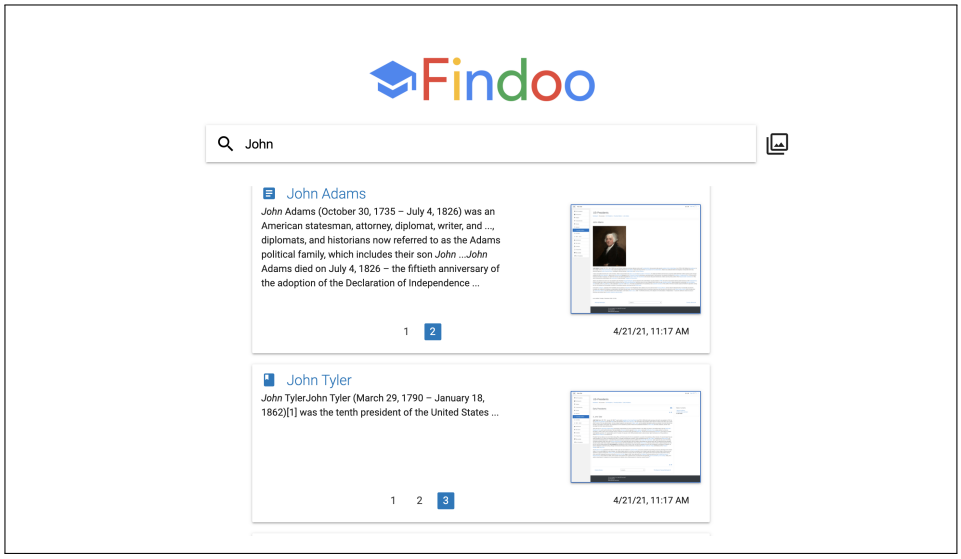


Abbildung 5.9: Die Standalone-Suche.

6 Evaluation

Dieses Kapitel beschreibt die Evaluation der Bestandteile des Systems. Zunächst wird die Funktionsweise des Deep Crawlers auf Korrekheit und Vollständigkeit geprüft. Dann werden die Performance der Suchmaschine, deren Subkomponenten und das Recommendersystem mit den in Kapitel 3 erklärten Metriken evaluiert. Dies findet via einer eigens konzipierten Evaluationsstrategie statt, die aus selbst erstellten Testumgebungen und Standard-Benchmarks besteht. Die einzelnen Komponenten des Systems werden dabei jeweils in Isolation betrachtet.

6.1 Die Testdaten und das Evaluationssetting

Für die Evaluation der Suchmaschine wurden vier verschiedene Datensätze verwendet, die an dieser Stelle genauer beschrieben werden.

- Als Beispiel für Realbedingungen wurde ein Datensatz synthetisiert, der tatsächliche E-Learning-Texte enthält. Er besteht aus 1028 Dokumenten, die jeweils einer von vier Kategorien zuordnen zu sind: Biologie, Physik, US-Geschichte und Geographie. Die Quellen dieser Texte sind:
  - Wikipedia, in der Regel der erste Abschnitt des jeweiligen Artikels
  - Die Biographien von US-Präsidenten auf der offiziellen Webseite des Weißen Hauses[1]
  - Blog-Einträge über Physik und Mathematik[2]
  - Artikel und Definitionen über Themen der Biologie[3]
- Als Beispiel für einen domänenspezifischen Datensatz wird der NF-Corpus [Bot+16] verwendet. Dieser besteht aus 3612 Dokumenten. Bei diesen handelt es sich um medizinische Fachdokumenten mit sehr technischen Themen und Formulierungen.
- Als zweiten domänenspezifischen Datensatz werden die *Collections of the ACM* (CACM) verwendet, bestehend aus 3205 Dokumenten. Bei diesen handelt es sich vor allem um Abstracts von Artikeln [SV88].

1 `http://whitehouse.gov/about-the-white-house/presidents`
2 `https://www.askamathematician.com/category/physicist/`
3 `https://biologydictionary.net`

- Als dritter domänenspezifischer Datensatz wird der Mikrobiologie-Datensatz des TREC 2007 Genomics Track, bestehend aus 2400 Dokumenten, verwendet [Her+06]. Auf diesem werden allerdings keine Benchmarks durchgeführt, da nur ein Bruchteil der gewählten Dokumente für die vorgegebenen Fragen relevant sind. Aufgrund der Komplexität der Dokumente und des Themas ist aber eine zum NF-Corpus ähnliche Performance zu erwarten.

Für die Evaluation des Crawlers wurden sechs Kurse in drei verschiedenen Learning Management Systemen verwendet:

- Drei Kurse über US-Geschichte, jeweils einer in Moodle, Ilias und Open edX
- Ein Kurs über Geographie in Moodle
- Ein Kurs über Mathematik in Moodle
- Ein Kurs ohne unterstützte Inhalte in Moodle

6.2 Evaluation des Crawlers

Im Folgenden wird der Crawler auf Korrekheit und Vollständigkeit geprüft. Die Experimente wurden in der oben beschriebenen Testumgebung durchgeführt. Konkret wurden ca. 250 Seiten betrachtet, davon enthalten 71 relevante Inhalte. Es ist zu beachten, dass ohne die in Kapitel 4 und Kapitel 5 diskutierte URL-Filterung durch die Blacklists und Whitelists die Zahl der Seiten um ungefähr eine Größenordnung höher wäre, die Kurse bestehen also aus sehr viel mehr Seiten. Der Ablauf des Experiments ist wie folgt: Es werden alle Tools durchsucht, die als relevant klassifizierten Inhalte werden indiziert und nach Beendigung der Suche mit den manuell bearbeiteten bzw. markierten idealen Ergebnissen verglichen.

Hypothese: Die indizierten Inhalte sind vollständig

Um die Nützlichkeit der Suchmaschine zu maximieren, muss der Crawling-Vorgang möglichst viele in den angeschlossenen Learning-Management-Systemen existierenden Inhalte finden und indizieren. Der hier verwendete Vollständigkeitsbegriff bezieht mit ein, dass nicht alle Inhalts-Typen unterstützt werden, sondern nur die oben erwähnten. Es ist ebenfalls zu beachten, dass nicht alle Inhalte atomar sind, die Sammlungs- und Quiz-Typen bestehen in der Regel aus mehreren Sub-Inhalten. An dieser Stelle werden diese alle isoliert betrachtet. Ob ein Inhalt indiziert wurde, wird mit der assoziierten URL bestimmt, welche Teile konkret gespeichert wurden wird im Rahmen der Korrektheit weiter unten untersucht.

Hypothese: Die indizierten Inhalte sind korrekt

Wie bereits erwähnt existiert eine konzeptionelle Lücke zwischen dem Inhalt (dem *Body*) und der Web-Seite, auf der er zu finden ist. Bei großen Teilen dieser handelt es sich um Markup und technische Elemente wie Navigationselemente etc. Der Body wird, wie in Kapitel 4 beschrieben, heuristisch bestimmt. Dieser Abschnitt der Arbeit evaluiert die Korrektheit bzw. Genauigkeit dieser Bestimmung. Für jeden relevanten Inhalt wurde der ideale Body manuell identifiziert und wird nun mit dem extrahierten Body verglichen. Als Metrik wird der Jaccard-Index verwendet. Dieser vergleicht die enthaltenen Token und gibt einen Wert im Intervall [0,1] als Gütemaß zurück. Der Wert 1 bedeutet, dass die Menge der Tokens identisch ist. Für einen falsch klassifizierten Inhalt ist dieser Jaccard-Index als 0 definiert. Es ist zu beachten, dass hierbei die Reihenfolge der Tokens nicht betrachtet wird, was aufgrund der Aufgabenstellung aber kein großes Problem darstellt - die Heuristik wird im Fehlerfall nur Abschnitte weglassen oder irrelevante hinzufügen, die Reihenfolge bleibt aber stets erhalten. Darüber hinaus ist sowohl für die Keyword-basierte Volltextsuche wie auch die semantische Suche die Reihenfolge außerhalb von Sätzen irrelevant.

6.3 Evaluation des Suchalgorithmus

Der Suchalgorithmus wird ebenfalls anhand einer Korrektheits- und Vollständigkeitshypothese bewertet. Neben diesen generellen Qualitätsmetriken ist zudem zu untersuchen wie sich die Komponenten des Algorithmus, namentlich der Text-Score, der Ontologie-Score und der Embedding-Score in Isolation und in Kombination verhalten. Am Ende wird eine dritte Hypothese untersucht, die besagt, dass die hier verwendete generische Form des Algorithmus in spezialisierten Domänen zu schlechteren Ergebnissen führt. Falls nicht explizit anders gesagt, bestehen die nachfolgenden Experimente immer daraus 30 Anfragen auf dem E-Learning-Datensatz durchzuführen und die Ergebnisse zu analysieren. Die Gewichte wurden dazu auf 1 für den TFIDF-Score, 2 für den Jaccard-Score und 4 für den Embedding-Score gesetzt. Darüber hinaus fand eine Normalisierung der Features wie in Kapitel 4 statt.

Hypothese: Die gefundenen Suchergebnisse sind vollständig

Ähnlich zur Vollständigkeitshypothese beim Crawler ist es wichtig, dass für ein gegebenes Informationsbedürfnis ein möglichst großer Teil der relevanten Dokumente zurückgegeben wird, ein Maß dafür ist der Recall. Dieser kann durch die Rückgabe aller Dokumente trivial maximiert werden. Diese Möglichkeit wird in diesem Kapitel ausgeschlossen, indem entweder

nur die ersten *k* Dokumente betrachtet werden oder der Recall in Abhängigkeit zur Korrektheit bzw. Präzision gesetzt wird.

Hypothese: Die gefundenen Suchergebnisse sind korrekt

Neben der Vollständigkeit ist die Korrektheit der Ergebnisse wichtig - von den zurückgegebenen Dokumenten sollen möglichst viele relevant sein. Auch hier gibt es eine einfache Lösung, nämlich das Zurückgeben von nur wenigen, vermeintlich einfach zu bewertenden Dokumenten. Dieses Problem wird ebenfalls durch das Betrachten der *precision@k* und der Kopplung an den Recall gelöst.

Hypothese: Das System funktioniert nur eingeschränkt in spezialisierten Domänen

Es ist abzusehen, dass die General-Purpose-Eigenschaft die Performance des Systems in spezialisierten Domänen beeinflussen kann bzw. wird. Hier ist nun zu prüfen wie viel schlechter sich das System verhalten wird, welche Komponenten davon wie stark betroffen sind und inwiefern der Ensemble-Ansatz diese Nachteile mitigieren kann. Hierzu werden die oben beschriebenen Benchmarks NF-Corpus und CACM verwendet und anhand der Mean Average Precision und des NDCG verglichen.

6.4 Evaluation des Recommendersystems

Für das Berechnen von inhaltlichen Vorschlägen gibt es die gleichen Anforderungen wie für die obigen zwei Anwendungsfälle. Das Recommender-System wird wie folgt evaluiert: Für 30 manuell erstellte Module werden Vorschläge berechnet und mit den erwarteten Vorschlägen verglichen, der Ablauf ist also (mit Ausnahme der Eingabe) äquivalent zur Evaluation der Suche. Da hier das Concept Source Problem allerdings weniger stark ausgeprägt ist (die Module sind ja bereits indiziert und annotiert), ist eine bessere Performance zu erwarten. Die Gewichte des Embedding Score und des Ontology Score werden beide auf 2 gesetzt. Eine Normalisierung findet nicht statt, da die Werte weniger *skewed* sind. Mit anderen Worten: Hier sind tatsächlich hohe Werte der Cosinus-Ähnlichkeit bzw. des Jaccard Index möglich und als Teil des Recommender-Algorithmus werden Inhalte mit Embedding Scores kleiner als ein Threshold (hier: 0.3) ohnehin nicht weiter betrachtet.

Hypothese: Die vorgeschlagenen Inhalte sind vollständig

Wenn es relevante Inhalte im Index gibt, sollten möglichst viele dieser auch vorgeschlagen werden. Es existiert hierbei die gleiche triviale Lösung wie oben (das Vorschlagen von allen Inhalten) und die gleiche Lösung wie oben (das Evaluieren der ersten k Vorschläge, sowie das Koppeln von Recall und Precision).

Hypothese: Die vorgeschlagenen Inhalte sind korrekt

Für die Korrektheitshypothese gilt das oben erwähnte ebenso wieder: Vorgeschlagene Inhalte müssen relevant sein und um triviale Lösungen auszuschließen wird wieder *precision@k* verwendet.

6.5 Ergebnisse und Diskussion

Diese Sektion beschreibt und diskutiert die Evaluationsergebnisse der Bestandteile des Assistenzsystems im Detail. Tabelle 6.1 gibt eine Kurzübersicht über die Evaluationsergebnisse der verschiedenen Hypothesen.

Evaluationsergebnisse: Crawler

Von den 71 relevanten Inhalten werden alle gefunden. Es wird dabei kein Inhalt falsch klassifiziert. Die Jaccard-Indices einer Stichprobe von 30 ausgewählten Inhalten sind sehr gut. Diese sind, wie in Tabelle 6.2 dargestellt, fast immer weit über 0.9. Einzig die Quiz-Seiten in Ilias sind mit 0.88 leichte Outlier. Dies lässt sich über deren komplexen Aufbau erklären, insbesondere die Vielzahl der interaktiven Fragen-Elemente erschwert die Identifizierung des Bodys.

Evaluationsergebnisse: Suche

Tabelle 6.3 gibt die Mean Average Precision und den NDCG der verschiedenen Komponenten des Suchalgorithmus an. Diese zeigt folgendes: Die Ontologie-Komponente hat durchwegs die schlechteste Performance. TFIDF und die Suche mit semantischen Einbettungen haben beide eine fast identische, durchschnittliche MAP, der NDCG-Wert von TFIDF ist allerdings recht deutlich besser. Die beiden Ensemble-Suchen haben die beste MAP-Performance, bieten also einen deutlichen Mehrwert. Auf Basis dieser Werte ist allerdings die Performance der beiden nahezu identisch, in der NDCG-Metrik kann sich die einschrittige Ensemble-Suche durchsetzen. Es ist aber zu beachten, dass hier die NDCG-Metrik für $k = 5$ gegeben ist. Sind mehr Dokumente relevant, hat der TFIDF-Algorithmus eine schlechtere Performance und einen entsprechend

Hypothese	Ergebnisse	Erklärung
Die indizierten Inhalte sind vollständig.	erfüllt	Von den 71 relevanten Inhalten werden alle gefunden, die Hypothese ist also vollständig erfüllt.
Die indizierten Inhalte sind korrekt.	erfüllt	Es werden keine Inhalte falsch klassifiziert, die Jaccard-Indizes sind alle weit über 0.9, einzig Ilias-Quizseiten sind mit 0.88 leichte Outlier. Die Hypothese ist also fast vollständig erfüllt.
Die Suchergebnisse sind korrekt und vollständig.	größtenteils erfüllt	Die Mean Average Precision der Ensemble-Suchen liegt bei 0.625 bzw. 0.622, der NDCG bei 0.773 bzw. 0.729 und die ROC-AUC bei 0.93 bzw. 0.94. Die Hypothese ist also größtenteils erfüllt.
Die Suche funktioniert nur eingeschränkt in spezialisierten Domänen.	erfüllt	Beim NF-Corpus beträgt die MAP der Ensemble-Suche nur 0.122 bzw. 0.116 und der NDCG 0.308 bzw. 0.295. Beim CACM-Benchmark ist die MAP 0.295 bzw. 0.289 und die NDCG 0.391 bzw. 0.379. Die Werte stellen aber eine leichte Verbesserung gegenüber dem TFDIF-Algorithmus dar. Die Hypothese ist also erfüllt, da insb. beim CACM-Benchmark die Performance der Suche nicht gut, aber akzeptabel ist.
Die vorgeschlagenen Inhalte sind korrekt und vollständig.	größtenteils erfüllt	Die Mean Average Precision des Recommender-Algorithmus beträgt 0.712 und die ROC-AUC ist 0.966. Die Hypothese ist also fast vollständig erfüllt.

Tabelle 6.1: Die Hypothesen und ihre experimentellen Ergebnisse.

Typ	Moodle	Ilias	edX
Page	0.98	0.93	0.94
Collection	0.99	0.98	-
Quiz	0.96	0.88	0.96
Exercise	0.95	0.93	0.95

Tabelle 6.2: Die Jaccard-Indices der verschiedenen Typen und LMS, dargestellt ist eine Stichprobe von 30 Inhalten. Diese sind durchgehend hoch, einzig Ilias-Quizelemente sind leichte Outlier.

Methode	MAP	NDCG
TFIDF	0.493	0.707
Embedding Search	0.489	0.601
Ontology Search	0.204	0.345
Ensemble Search (a)	0.625	0.773
Ensemble Search (b)	0.622	0.729

Tabelle 6.3: Mean Average Precision und NDCG der einzelnen Komponenten. Die Ensemble-Suchen bieten einen Mehrwert, TFIDF hat aber auch recht hohe Werte.

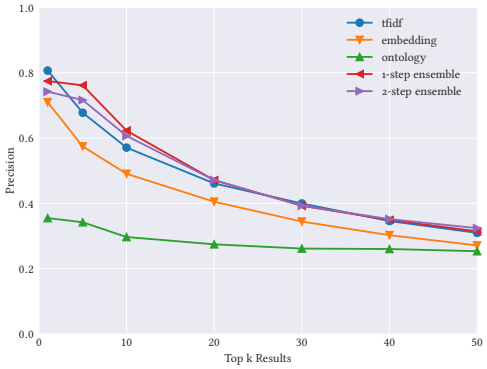
Methode	TFIDF	Embeddings	Ontologie	Ensemble (a)	Ensemble (b)
ROC-AUC	0.838	0.887	0.714	0.93	0.946

Tabelle 6.4: Die ROC-AUC-Metrik für die verschiedenen Methoden, die Ensemble-Methoden sind deutlich überlegen.

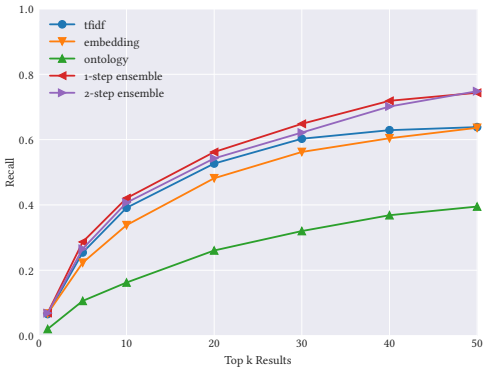
niedrigeren Wert, die Ensemble-Methoden sind davon weniger stark betroffen. Abbildung 6.1 zeigt die Präzision, den Recall sowie den F1-Score des Systems und dessen Komponenten in Abhängigkeit von den ersten k Dokumenten, die zurückgegeben werden. Dies zeigt folgendes: Wird nur das erste Dokument betrachtet, hat TFIDF einen leichten Vorteil in der Präzision, bis $k = 20$ sind die Ensemble-Methoden besser und danach sind alle drei vergleichbar. Beim Recall können sich die Ensemble-Methoden allerdings eindeutig durchsetzen, was auch dazu führt, dass sie einen besseren F1-Score vorweisen. Es gilt also: Wenn Elemente via TFIDF gefunden werden, sind diese in der Regel auch relevant, es werden aber einige gar nicht gefunden. Dies deckt sich mit der Intuition hinter dem Konzept der syntaktischen Suche. Wie am F1-Score zu sehen ist, scheinen sich die Vorteile der beiden Methoden aber etwas auszugleichen. Konsistent mit den obigen Metriken ist die Ontologie-Suche eindeutig schlechter, die Ensemble-Suche ist zwischen dieser und den anderen Ansätzen einzuordnen.

Dass diese Ergebnisse (bedingt durch k) aber etwas irreführend sind zeigt Abbildung 6.2. Hier sind die Ensemble-Methoden deutlicher überlegen. Für ein Recall-Level von 0.8 zum Beispiel hat die zweischrittige Ensemble Search eine Präzision von über 0.4, TFIDF hat eine von 0.26. Die Fläche unter den ROC-Kurven (genannt die ROC-AUC-Metrik) ist dargestellt in Tabelle 6.4. Hier ist zu sehen, dass die Ensemble-Methoden um 0.1 bis 0.11 besser sind. Abschließend wird die kategorische Präzision betrachtet. Diese ist dargestellt in Abbildung 6.3 und ist durchgehend sehr hoch, hier haben die Ensemble-Methoden ebenfalls Vorteile.

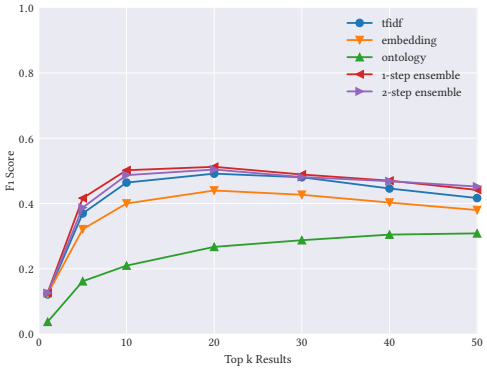
Bei genauerer Betrachtung scheinen die Graphen in Abbildung 6.2 inkonsistent zu sein. Der Precsision-Recall-Graph zeigt die einschrittige Ensemble-Suche als leicht besser, die ROC-



(a) Präzision

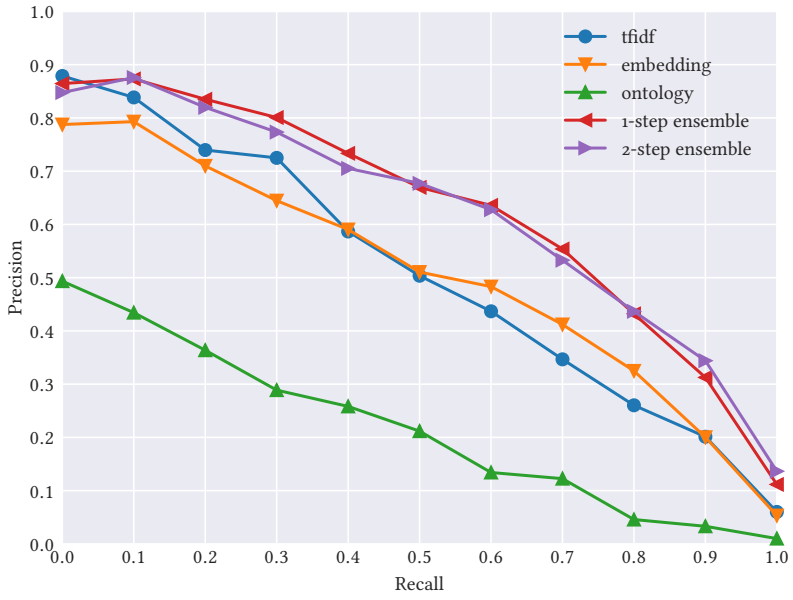


(b) Recall

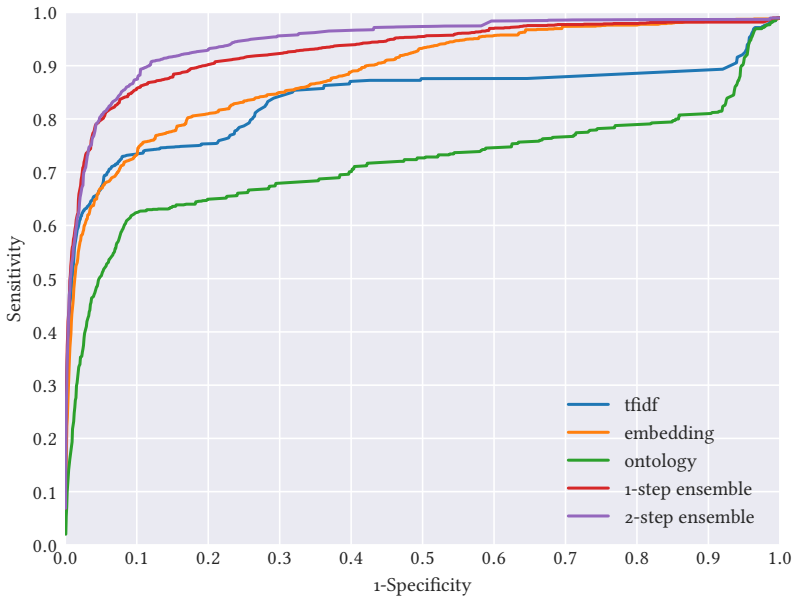


(c) F1-Score

Abbildung 6.1: Precision, Recall und F1-Score des Suchalgorithmus abhängig von den ersten k Ergebnissen. Dargestellt sind Durchschnittswerte aus 30 Anfragen. Die Ensemble-Methoden haben einen klar besseren Recall, sind bei der Präzision allerdings mit TFIDF vergleichbar.



(a) Precision-Recall



(b) ROC

Abbildung 6.2: Interpolierte Precision-Recall-Graphen und ROC-Graphen. Dargestellt sind Durchschnittswerte aus 30 Anfragen. Beide Graphen zeigen die Überlegenheit der Ensemble-Methoden und die stark limitierte Performance der Ontologie-Suche.

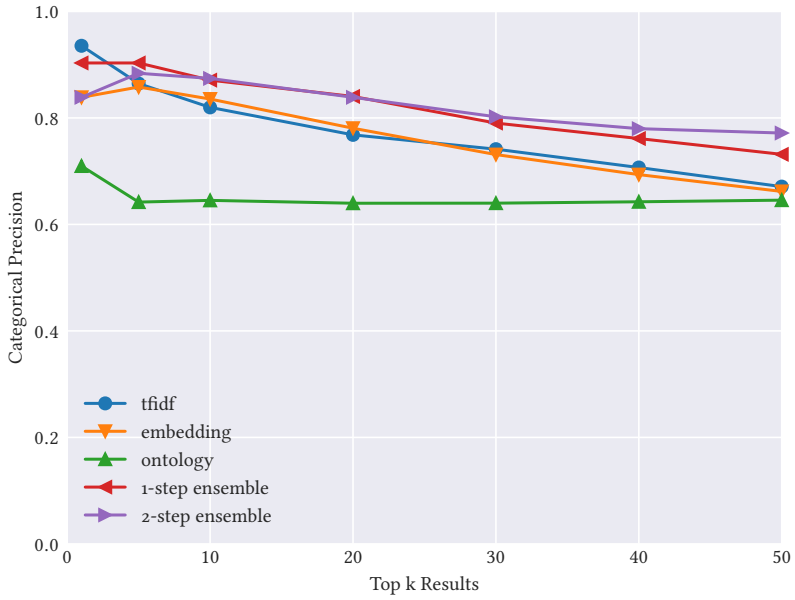


Abbildung 6.3: Die kategorische Präzision des Suchalgorithmus abhängig von den ersten k Ergebnissen. Dargestellt sind Durchschnittswerte aus 30 Anfragen. Die Präzision ist durchweg sehr hoch, die Ensemble-Methoden sind allerdings ab $k = 5$ deutlich überlegen.

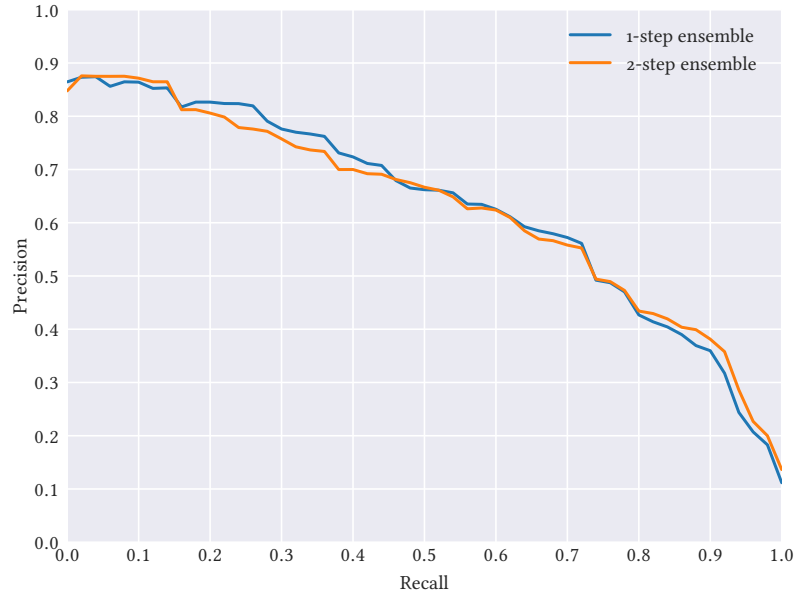


Abbildung 6.4: Die beiden Ensemble-Methoden im detaillierten Vergleich. Der genaue Kurvenverlauf weist eine Oszillation auf, keine Methode ist eindeutig überlegen.

Kurve die zweischrittige. Dies wäre im Widerspruch zu dem von Davis und Goedrich [DG06] formulierten Theorem, dass eine ROC-Kurve genau dann eine andere dominiert, wenn die entsprechenden Precision-Recall-Kurven sich dominieren. Die hier erzielten Ergebnisse stellen allerdings keinen solchen Widerspruch da, denn bei genauerer Betrachtung der Kurven mit 51 statt 11 Stützpunkten (Abbildung 6.4) ist eine starke Oszillation zu erkennen, hier betragen die Flächen unter der Precision-Recall-Kurve 0.633 bzw. 0.63. Das Theorem ist also nicht verletzt, da (wie im Paper erwähnt) die lineare Interpolation im Precision-Recall-Graphen immer einen Fehler impliziert. Dieser wird bei nur sehr geringen Performanceunterschieden umso kritischer. Außerdem ist zu beachten, dass es sich hier um Aggregationsplots von 30 Kurven handelt, die Anwendbarkeit des Theorems also generell unklar ist.

Wie oben beschrieben bestehen die Ensemble-Methoden jeweils aus den drei Komponenten der TFIDF-, Einbettungs- und Ontologie-Suchen. Abbildung 6.5 stellt die Precision-Recall-Graphen der möglichen zweielementigen Kombinationen dar und zeigt, dass die tatsächliche Kombination aller drei Komponenten einen echten Mehrwert bietet, jedoch von den Beiträgen der Einbettung und des TFIDF-Algorithmus dominiert wird. Die Kombination aus TFIDF und Ontologien hat eine MAP von 0.39 (NDCG: 0.56) und ist damit schlechter als die TFIDF-Suche allein, Einbettungen und Ontologien haben eine MAP von 0.52 (NDCG: 0.62) und TFIDF in Kombination mit Einbettungen hat eine MAP von 0.6 (NDCG: 0.74).

Wird das System auf einen spezialisierten Anwendungsfall angewendet, geht die Performance zurück, jedoch nicht unbedingt auf katastrophale Weise. Wie die Tabellen 6.5 und 6.6 belegen, ist die Performance für den CACM-Datensatz sehr viel schlechter, aber noch akzeptabel. Für den NF-Corpus ist das System aber nicht anwendbar. Hier gilt es aber auch zwischen den einzelnen Komponenten zu unterscheiden. Der fundamentale Unterschied zwischen dem allgemeinen und dem spezifischen Anwendungsfall besteht darin, dass im ersten Fall aus vielen verschiedenen Dokumenten ein paar spezifische gefunden werden müssen, und im zweiten aus sehr ähnlichen Dokumenten ein paar spezifische gesucht werden. Die Ontologie-Performance ist ein illustratives Beispiel. Wenn nur wenige Dokumente mit bestimmten Konzepten annotiert sind, sind diese aussagekräftig. Wenn allerdings fast alle Dokumente mit diesen annotiert werden, geht jegliche Information verloren. Beim NF-Corpus ist der Embedding-Ansatz von einem ähnlichen Problemm betroffen, was sich mit der Arbeit von Frei [FSC20] deckt, dort hatte ein untrainiertes BERT-Modell, was hier ja auch vorliegt, eine MAP von 0.05, hier sind es 0.048. Die Erklärung ist durch die Funktionswiese des Embedding-Ansatzes gegeben, dort findet aus konzeptioneller Sicht ein hierarchisches Clustering von Vektoren in einem 768-dimensionalen Raum statt. Sind die Elemente der Vektoren nicht feingranular genug, ist das Clustering auch nicht fein genug. Diese repräsentieren dann beispielsweise alle das Thema der Medizin und nicht die Details verschiedener Subdisziplinen. Der TFIDF-Algorithmus ist robuster als die

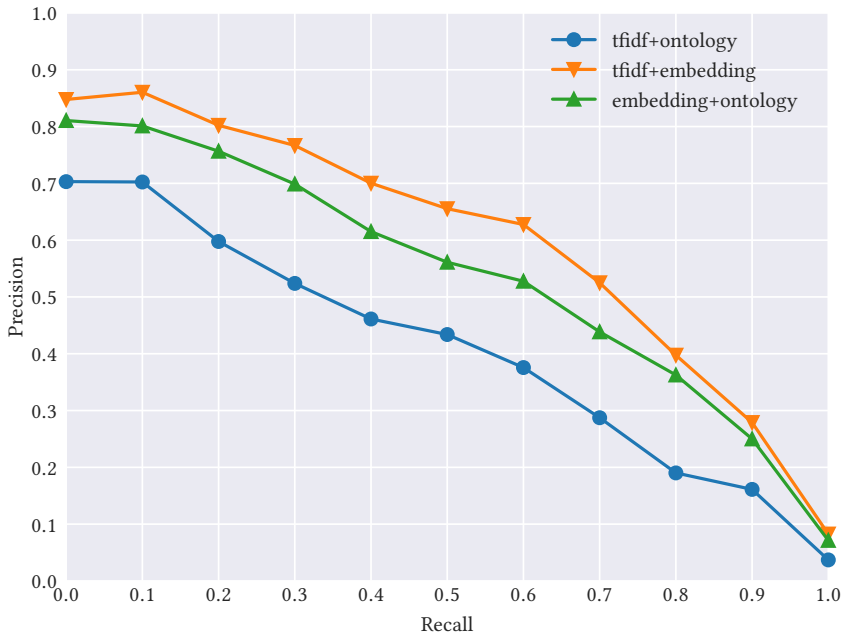


Abbildung 6.5: Interpolierte Precision-Recall-Graphen für die möglichen alternativen Kombinationen der Such-Komponenten. Die Kombination aus TFIDF und Einbettungen ist überlegen.

anderen beiden Komponenten, was sich in der durchaus nicht zu verachtenden Performance, insb. beim CACM-Benchmark zeigt. Ebenfalls interessant ist, dass bei diesem Benchmark der einschrittige Ensemble-Ansatz etwas besser ist als der zweischrittige. Desweiteren ist die Performance von den eigentlichen Eingaben abhängig - das System ist für Keyword-basierte Suchen entworfen und nicht für längere natürlichsprachigere Eingaben, wie sie in den Benchmarks verwendet werden. Hierfür müsste ein Question-Answering-Modell verwendet werden. Auch die Länge der indizierten Dokumente kann eine Rolle spielen - sind diese zu kurz können im Rahmen der Ontologie nur wenige Konzepte erkannt werden. Generell gilt für das System aber: *do no harm*. Die semantische Suche ist deaktivierbar und TFIDF kann in der Standard-Implementierung genutzt werden, ebenso ist es möglich die Gewichte der einzelnen Komponenten anzupassen. Wie diese Evaluation zeigt, ist dessen Performance in Fällen von nur wenigen relevanten Dokumenten auch gut. Für zukünftige Arbeiten ist zudem insbesondere die modulare Struktur relevant. Für die semantischen Einbettungen und die Ontologie können beliebige Module eingesetzt werden. Erste Experimente zeigen aber, dass das einfache

Benchmark	TFIDF	Embeddings	Ontologie	Ensemble (a)	Ensemble (b)
E-Learning	0.493	0.489	0.204	0.625	0.622
NF-Corpus	0.108	0.048	0.046	0.122	0.116
CACM	0.289	0.051	0.005	0.295	0.289

Tabelle 6.5: Mean Average Precision der Suchalgorithmen in verschiedenen Benchmarks. Alle Methoden sind auf den spezifischen Datensätzen schlechter.

Benchmark	TFIDF	Embeddings	Ontologie	Ensemble (a)	Ensemble (b)
E-Learning	0.707	0.601	0.345	0.773	0.729
NF-Corpus	0.335	0.146	0.151	0.308	0.295
CACM	0.384	0.107	0.019	0.391	0.379

Tabelle 6.6: Normalized Discounted Cumulative Gain der Suchalgorithmen in verschiedenen Benchmarks für $k = 5$. Alle Methoden sind auf den spezifischen Datensätzen schlechter.

Austauschen von Modellen deren Training nicht adäquat ersetzen kann. Beispielsweise ist die Performance des Systems im NF-Corpus-Benchmark mit Bio-BERT, einem BERT-Modell, dass auf biologischen Daten trainiert worden ist, sogar leicht schlechter - eine erklärende Hypothese hierbei wäre, dass die gewonnene Verfeinerung der Einbettungen dadurch mitigiert wird, dass das Modell weniger gut mit ganzen Sätzen oder deren Aggregation umgehen kann. Auf technischer Ebene kann auch der Ursprung der Einbettungen modifiziert werden - bisher stammen diese aus der letzten Ebene, es gibt aber auch Ansätze diese aus früheren Ebenen zu verwenden, insbesondere wenn das Modell nicht mehr weiter trainiert wird. Dies würde allerdings wiederum der Sentence-BERT-Hypothese [RG19] widersprechen.

Evaluationsergebnisse: Recommendersystem

Bedingt durch das Design des Recommender-Algorithmus kann dieser als eine Ausprägung des oben behandelten Suchproblems betrachtet werden. Es ist zunächst allerdings unklar, ob es sich dabei um eine Vereinfachung oder ein erschwertes Problem handelt. Wenn die Eingabe nicht nur aus einem Suchterm, sondern aus Dokumenten besteht ist das Ableiten von Konzepten und semantischer Einbettungen trivial - diese sind ja bereits gegeben. Allerdings müssen die gesuchten Dokumente eine höhere Relevanzgrenze überschreiten, um aktiv vorgeschlagen zu werden. Im Allgemeinen sind also für ein gegebenes Thema weniger Dokumente geeignete Vorschläge als für Suchen bezüglich Begriffen zu diesem Thema gefunden werden sollen. Die Ergebnisse zeigen allerdings, dass die Aufgabe tatsächlich besser gelöst werden kann als die Suche. Das

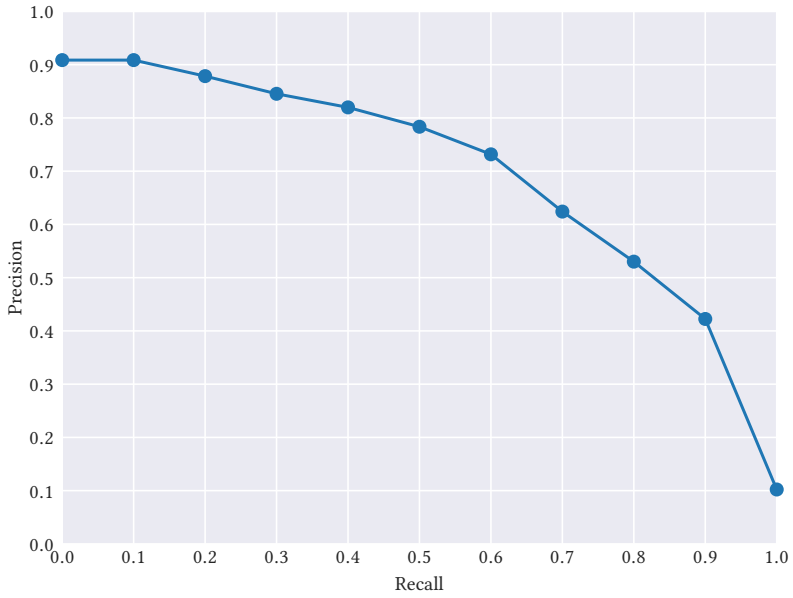


Abbildung 6.6: Der interpolierte Precision-Recall-Graph des Recommender-Systems. Die Performance ist leicht besser als die des Suchalgorithmus.

System hat eine Mean Average Precision von 0.712 und eine ROC-AUC von 0.96 auf dem E-Learning-Datensatz. Wie Abbildung 6.6 darstellt sind Präzision und Recall durchweg höher als beim Suchproblem. Abbildung 6.9 illustriert folgendes: Der Recall@k des Systems ist sehr viel höher beim Suchproblem, die Präzision@k allerdings nicht. Dies hat aber nur den Grund, dass für die meisten getesteten Module höchsten 15 Inhalte relevant waren. Die Betrachtung von k Elementen ist nicht stabil in dieser Hinsicht. Die kategorische Präzision ist in Abbildung 6.7 dargestellt und zeigt, dass für einen weniger strengen Relevanzbegriff eine sehr gute Präzision erreicht wird - vorgeschlagene Dokumente sind also in nahezu allen Fällen zumindest dem gleichen übergreifenden Thema (z.B. *Biologie*) zuzuordnen. Abbildung 6.8 stellt schließlich die Precision-Recall-Graphen der beiden Subkomponenten des Recommender-Systems dar, die im eigentlichen Algorithmus wieder in einem Ensemble kombiniert werden. Der Mehrwert dessen ist hier noch deutlicher zu sehen als beim Such-Algorithmus. Die Vorschläge des Ontology-Scores haben eine MAP von 0.24, die der Einbettungen 0.57. Das Ensemble hat wie oben gezeigt eine MAP von 0.71 - die beiden Werte addieren sich also zu einem signifikanten Grad. Dies lässt darauf schließen, dass die Vorschläge der beiden Subkomponenten oft orthogonal zueinander sind, sich also recht gut ergänzen.

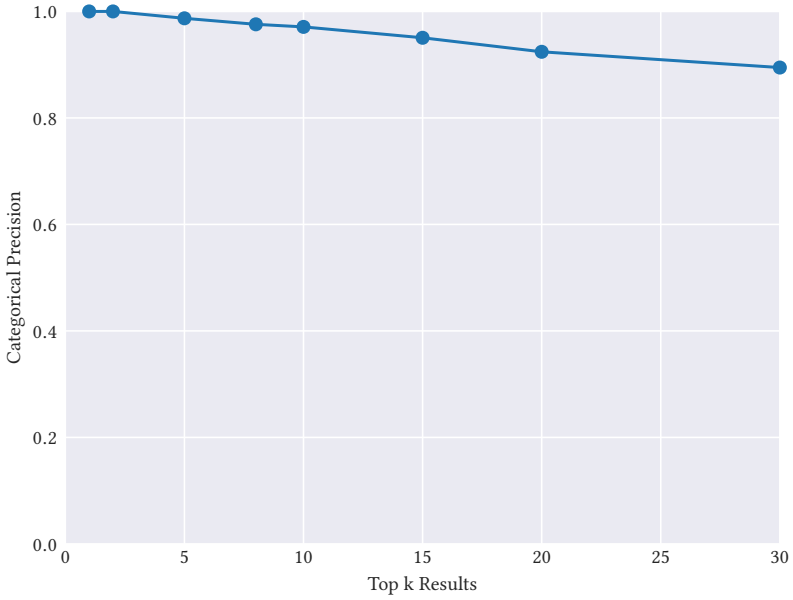


Abbildung 6.7: Die kategorische Präzision des Recommender-Algorithmus abhängig von den ersten k Ergebnissen. Diese ist selbst für große k sehr hoch.

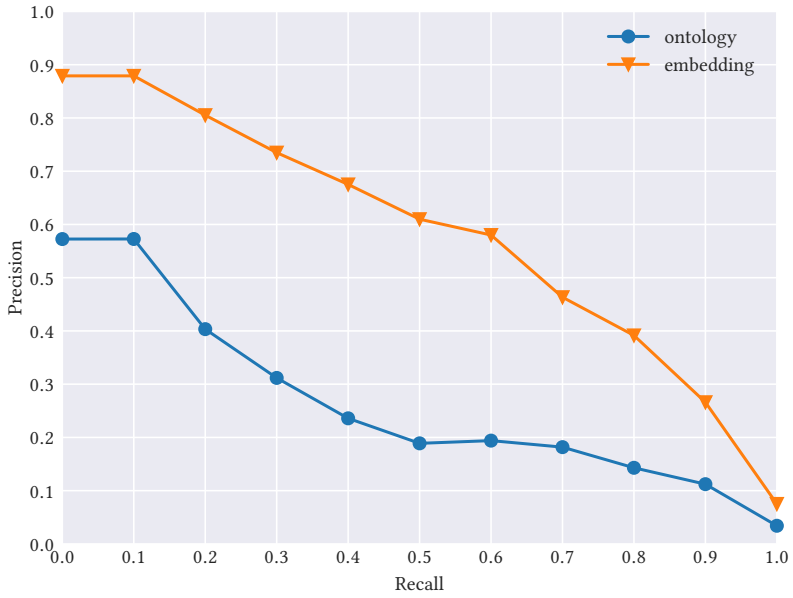
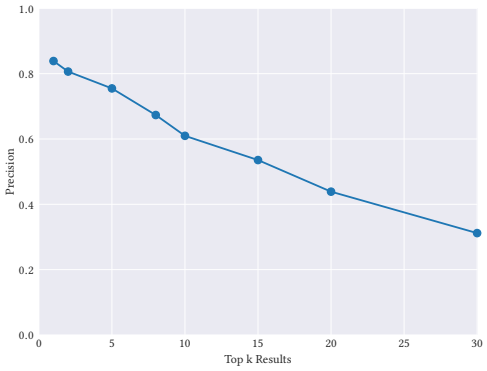
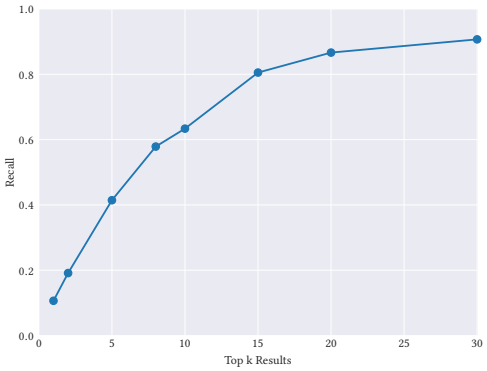


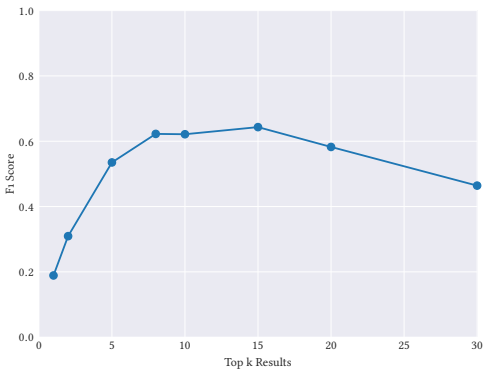
Abbildung 6.8: Der interpolierte Precision-Recall-Graph der beiden Subkomponenten des Recommender-Systems. Die Einbettungs-Methode dominiert klar.



(a) Präzision



(b) Recall



(c) F1-Score

Abbildung 6.9: Precision, Recall und F1-Score des Recommender-Algorithmus abhängig von den ersten k Ergebnissen. Dargestellt sind Durchschnittswerte aus 30 Anfragen. Der Recall ist für große k sehr gut, die Präzision ist vergleichbar mit der des Suchalgorithmus.

7 Fazit und Ausblick

Mit konventionellen Techniken ist es Suchmaschinen nicht möglich E-Learning-Inhalte in Learning Management Systemen zu indizieren. Die im Rahmen dieser Arbeit untersuchte Forschungsfrage bezieht sich auf die Lösung dieses Auffindbarkeitsproblems durch die Anwendung von Interoperabilitätstechniken und E-Learning-Standards wie Learning Tools Interoperability (LTI) [IMS19a].

Die zentralen Beiträge dieser Arbeit sind sowohl praktischer als auch theoretischer Natur. Es wurde eine Suchmaschine und ein Kurationssystem für E-Learning-Inhalte entwickelt, das in die am Fraunhofer IOSB entwickelte Common Learning Middleware (CLM) [IOS21] eingegliedert ist. Bei den konzeptionellen Aspekten, die für den Entwurf dieses Systems grundlegend sind, handelt es sich um Themen der Interoperabilität von Softwaresystemen sowie der Entwicklung eines Deep Web-Crawlers und Information Retrieval- sowie Recommender-Algorithmen. Neben den theoretischen Faktoren muss bei der Entwicklung des Systems auch stets der technische Kontext und die Kooperation mit anderen Organisationen und Teams beachtet werden. In der Literatur gibt es kein vergleichbares System, deswegen werden die Ergebnisse gemäß eines eigenen Evaluationskonzeptes bewertet. Dieses betrachtet die einzelnen Teile des Systems in Isolation und verwendet sowohl eigens entworfene Testdatensätze als auch Standard-Benchmarks.

Im Bereich der Interoperabilität wird die Anwendbarkeit von existierenden Standards sowie deren Limitierungen und Lösungen für diese diskutiert. Die Erkenntnisse zu diesem Thema werden in Form einer Reihe von Regeln und Best Practices zusammengefasst. Diese haben ihre Wurzeln in traditionellem Software-Design, der neue Kontext führt aber zu neuen Herausforderungen. Ebenfalls hierzu gehört das Thema der Robustheit, diese wird an einem robusten, LTI-basierten Web-Crawling-Algorithmus für E-Learning-Inhalte in Learning Management Systemen gezeigt. Dieser muss mit unbekannten Umgebungen und potentiellen Änderungen an den Systemen umgehen können. Für den Algorithmus wurden Features wie Near Duplicate Detection für zu indizierende Texte, die Klassifikation von Inhalten anhand von Ankerpunkten, CSS-Abstraktion und die systemunabhängige Find-Body-Heuristik zur Erkennung des natürlichsprachlichen Text auf Webseiten entworfen. Als unmittelbare Folge der Implementierung des Crawlers erfolgt zudem eine automatisierte Metadatensynthese. Hiermit wird die Erfüllung

eines operationellen Robustheitsbegriffes verfolgt: Das System muss ohne Aufwand mit kleinen Änderungen umgehen können und leicht an große Änderungen anpassbar sein.

Für die Information-Retrieval-Komponente des Systems müssen Probleme bezüglich des Designs eines suchbaren Indexes und den dort zu speichernden Features gelöst werden. Auf diesem wird zunächst eine Volltextsuche implementiert. Diese wird weiter um eine semantischen Suche ergänzt, hierfür werden eine Reihe von Techniken des Natural Language Processing angewendet, insbesondere semantische Einbettungen und Ontologien. Der entworfene Algorithmus ist dabei in der Lage gleichzeitig als Suchmaschine und Recommender-System verwendet zu werden. Ein zentrales Problem der semantischen Suche ist jedoch, dass die verwendeten Modelle im Kontrast zu konventionellen Machine Learning-Aufgaben nach dem Pre-Training nicht auf repräsentativeren verfeinert werden können, da die Domäne der Inhalte nicht a-priori bekannt ist.

Die Auswirkungen dieser Anforderung werden zum Abschluss der Arbeit diskutiert, hier werden die einzelnen Komponenten des System isoliert und in Kombination systematisch anhand einer Reihe von Standard-Metriken evaluiert. Diese zeigen eine hohe Korrektheit des Crawling-Algorithmus. Dieser ist in der Lage alle gewünschten Inhalte zu finden und diese mit sehr hoher Präzision zu indizieren: Die Mengen der tatsächlich indizierten Tokens und die vorgegebenen, perfekten Ergebnisse haben stets einen Jaccard-Index von weit über 0.9. Die Evaluation zeigt ebenso eine gute Performance des Information Retrieval-Systems auf einem synthetisierten E-Learning-Datensatz: Eine Mean Average Precision von 0.625 und einen Normalized Discounted Cumulative Gain von 0.773. Die Ergebnisse für das Recommender-System verhalten sich ähnlich, dieses hat eine Mean Average Precision von 0.71.

Die Evaluation zeigt aber auch Limitierungen und potentielle Weiterentwicklungen im Bereich der semantischen Suchfunktionalität. Bedingt durch die General-Purpose-Anforderung kann das System nicht in vollem Umfang auf die Anwendung in spezifischen Domänen vorbereitet werden, hier wird eine maximale Mean Average Precision von 0.295 festgestellt. Doch gerade weil diese Limitierung in der Aufgabenstellung inhärent ist wurde eine Reihe von Designentscheidungen getroffen, die die entstehenden Probleme mitigieren. Die erste dieser Entscheidungen betrifft das *Do no harm*-Prinzip: Die Suchmaschine enthält, bedingt durch die zur Implementierung verwendeten Technologien, eine state-of-the-art Implementierung des TFIDF-Suchalgorithmus, dieser ist stabil gegenüber der dem Text-Corpus unterliegenden Domäne und stellt die Funktionsweise des Systems sicher. Ist es gewünscht die semantische Suchfunktionalität zu verbessern, kann der modulare Aufbau des Algorithmus genutzt werden. Dieser besteht aus Komponenten, die prinzipiell über einen beliebigen internen Aufbau verfügen und flexibel kombiniert werden können. In der Arbeit werden ein Minimal-Modell und eine Variante vorgestellt, diese zeigen einen ersten Ansatz.

Ausblick

Das entworfene System ist auf Erweiterbarkeit und Adaptierbarkeit ausgelegt. Eine naheliegende Erweiterung ist das Hinzufügen von neuen Crawlern für andere Learning Management Systeme oder Inhaltstypen, dies ist softwaretechnisch mit geringem Aufwand möglich. Auf technischer Seite kann die Migration auf mehrere bzw. andere Standards wie LTI 1.3 [IMS19b] untersucht werden, dies kann auch in Co-Evolution mit der Common Learning Middleware geschehen. Ein weiterer naheliegender Forschungs-Aspekt ist die Aufgabe der General-Purpose-Anforderung und die Anwendung des entworfenen Information Retrieval-Algorithmus auf spezielle Anwendungsdomänen. Aufgrund dessen modularen Aufbaus ist die technische Umsetzung von etwaigen Lösungen nicht schwer, wie erste Experimente zeigen muss aber in der Modellfindung noch theoretische Arbeit geleistet werden. Für weiterführende Suchaufgaben wie die des Question-Answering kann die Arbeit ebenfalls als Grundlage gesehen werden. Gleiches gilt für das Recommender-System, dessen Weiterenticklung ist allerdings noch stärker von der Betriebsumgebung und den dort bereitgestellten Kontextinformationen zu Nutzern oder Inhalten abhängig, hier muss entweder die CLM erweitert oder auf eine andere Umgebung umgestiegen werden.

Literatur

[AB17] Norah Aldaij und Jawad Berri. „E-Learning Authoring Tool for Reusing Web Multimedia Resources“. In: *E-Learning, E-Education, and Online Training*. Hrsg. von Giovanni Vincenti u. a. Bd. 180. Series Title: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Cham: Springer International Publishing, 2017, S. 153–160. URL: http://link.springer.com/10.1007/978-3-319-49625-2_19 (besucht am 12. 11. 2020).

[ADL09] ADL. *SCORM 2004 4th edition*. 2009.

[Ado21] Adobe. *Adobe Captivate*. 2021. URL: https://www.adobe.com/de/products/captivate.html (besucht am 20. 04. 2021).

[AF06] Shaaron Ainsworth und Piers Fleming. „Evaluating authoring tools for teachers as instructional designers“. In: *Computers in Human Behavior* 22.1 (Jan. 2006), S. 131–148. (Besucht am 02. 12. 2020).

[AHH12] Ali Alharbi, Frans Henskens und Michael Hannaford. *A Domain-Based Learning Object Search Engine to Support Self-Regulated Learning*. 2012.

[Ahn+17] Jeong Yong Ahn u. a. „An online authoring tool for creating activity-based learning objects“. In: *Education and Information Technologies* 22.6 (Nov. 2017), S. 3005–3015. URL: http://link.springer.com/10.1007/s10639-016-9567-9 (besucht am 12. 11. 2020).

[AIS93] Rakesh Agrawal, Tomasz Imielinski und Arun Swami. „Mining association rules between sets of items in large databases“. In: *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. 1993, S. 207–216.

[Ale+06] Vincent Aleven u. a. *The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains*. 2006.

[Ale+16] Vincent Aleven u. a. „Embedding Intelligent Tutoring Systems in MOOCs and e-Learning Platforms“. In: *Intelligent Tutoring Systems*. Hrsg. von Alessandro Micarelli, John Stamper und Kitty Panourgia. Bd. 9684. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, S. 409–415.

URL: http://link.springer.com/10.1007/978-3-319-39583-8_49 (besucht am 12. 11. 2020).

[AT11] Kohei Arai und Herman Tolle. *Efficiency Improvement of E-Learning Document Search Engine for Mobile Browser.* 2011.

[AZ08] Ajlan Al-Ajlan und Hussein Zedan. „Why moodle". In: *2008 12th IEEE International Workshop on Future Trends of Distributed Computing Systems.* IEEE. 2008, S. 58–64.

[Bak+17] Abdellah Bakhouyi u. a. „Evolution of standardization and interoperability on E-learning systems: An overview". In: *2017 16th International Conference on Information Technology Based Higher Education and Training (ITHET).* 2017 16th International Conference on Information Technology-Based Higher Education and Training (ITHET). Ohrid: IEEE, Juli 2017, S. 1–8. URL: https://ieeexplore.ieee.org/document/8067789/ (besucht am 23. 11. 2020).

[Bat08] Paulo Eduardo Battistella. *Evaluation of Free Authoring Tools for producing Learning Objects on SCORM.* 2008.

[Bed+13] S Bednar u. a. „Adoption of ILIAS Web Learning System for distance education". In: *2013 IEEE 11th International Conference on Emerging eLearning Technologies and Applications (ICETA).* IEEE. 2013, S. 139–144.

[Ber01] Michael K Bergman. „White paper: the deep web: surfacing hidden value". In: *Journal of electronic publishing* 7.1 (2001).

[BKM91] Nigel Bevana, Jurek Kirakowskib und Jonathan Maissela. „What is usability". In: *Proceedings of the 4th International Conference on HCI.* Citeseer. 1991.

[Bot+16] Vera Boteva u. a. „A Full-Text Learning to Rank Dataset for Medical Information Retrieval". In: 2016. URL: http://www.cl.uni-heidelberg.de/~riezler/publications/papers/ECIR2016.pdf.

[BP98] Sergey Brin und Lawrence Page. „The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN Systems* 30.1 (Apr. 1998), S. 107–117.

[BWB09] Yevgen Biletskiy, Michael Wojcenovic und Hamidreza Baghi. *Focused Crawling for Downloading Learning Objects – An Architectural Perspective.* 2009.

[ÇA18] Mehmet Ali Çinici und Arif Altun. „Reusable content matters: a learning object authoring tool for smart learning environments". In: *Smart Learning Environments* 5.1 (Dez. 2018), S. 10. URL: https://slejournal.springeropen.com/articles/10.1186/s40561-018-0060-3 (besucht am 12. 11. 2020).

[CD99] Soumen Chakrabarti und Byron Dom. *Focused Crawling: A New Approach to Topic-Specific Resource Discovery*. 1999.

[Con21] IMS Global Consortium. *IMS Initiatives*. 2021. URL: https://www.imsglobal.org/institutions.html (besucht am 02. 05. 2021).

[CT+94] William B Cavnar, John M Trenkle u. a. „N-gram-based text categorization". In: *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*. Bd. 161175. 1994.

[Dai+13] Joachim Daiber u. a. „Improving efficiency and accuracy in multilingual entity extraction". In: *Proceedings of the 9th International Conference on Semantic Systems - I-SEMANTICS '13*. the 9th International Conference. Graz, Austria: ACM Press, 2013, S. 121. URL: http://dl.acm.org/citation.cfm?doid=2506182.2506198 (besucht am 10. 04. 2021).

[Dai12] Robert Daigneau. *Service Design Patterns: fundamental design solutions for SOAP / WSDL and restful Web Services*. Addison-Wesley, 2012.

[DBp21] DBpedia. *DBpedia Lookup*. 2021. URL: https://github.com/dbpedia/dbpedia-lookup (besucht am 06. 05. 2021).

[Del+20] Daniele Dellagiacoma u. a. „Authoring Interactive Videos for e-Learning: The ELEVATE Tool Suite". In: *Methodologies and Intelligent Systems for Technology Enhanced Learning, 10th International Conference*. Hrsg. von Pierpaolo Vittorini u. a. Bd. 1241. Series Title: Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2020, S. 127–136. URL: http://link.springer.com/10.1007/978-3-030-52538-5_14 (besucht am 12. 11. 2020).

[Dev+19] Jacob Devlin u. a. „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv:1810.04805 [cs]* (24. Mai 2019). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (besucht am 27. 03. 2021).

[DG06] Jesse Davis und Mark Goadrich. „The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, S. 233–240.

[DG13] Manda Sai Divya und Shiv Kumar Goyal. „ElasticSearch: An advanced and quick search technique to handle voluminous data". In: *Compusoft* 2.6 (2013), S. 171.

[Dow05] Stephen Downes. „E-learning 2.0". In: *eLearn* 2005.10 (2005), S. 1.

[edu13] edu-apps.org. *Writing LTI Stuff*. 2013. URL: https://www.edu-apps.org/code.html#intro (besucht am 02. 05. 2021).

[eli19] eliterate.us. *State of Higher Ed LMS Market for US and Canada: 2018 Year-End Edition.* 2019.

[EO19] Kamal El Guemmat und Sara Ouahabi. „Towards a new educational search engine based on hybrid searching and indexing techniques". In: *2019 1st International Conference on Smart Systems and Data Science (ICSSD).* 2019 1st International Conference on Smart Systems and Data Science (ICSSD). Rabat, Morocco: IEEE, Okt. 2019, S. 1–5. URL: https://ieeexplore.ieee.org/document/9002729/ (besucht am 23. 04. 2021).

[ET18] Kamal El Guemmat und The Society of Digital Information and Wireless Communication. „Issues of E-Learning Search Engine and its Challenges". In: *The International Journal of E-Learning and Educational Technologies in the Digital Media* 4.3 (2018), S. 75–85. URL: http://sdiwc.net/digital-library/issues-of-elearning-search-engine-and-its-challenges.html (besucht am 23. 04. 2021).

[eV21a] Ilias open source e-Learning e.V. *Ilias Demo.* 2021. URL: https://demo.ilias.de/ (besucht am 05. 05. 2021).

[eV21b] Ilias open source e-Learning e.V. *Ilias Development Guide.* 2021. URL: https://docu.ilias.de/goto_docu_pg_29964_42.html (besucht am 05. 05. 2021).

[FDA19] B. Faqihi, N. Daoudi und R. Ajhoun. „Proposition of the recommendation system for the author based on similarity degrees". In: *2019 1st International Conference on Smart Systems and Data Science (ICSSD).* 2019 1st International Conference on Smart Systems and Data Science (ICSSD). Rabat, Morocco: IEEE, Okt. 2019, S. 1–7. URL: https://ieeexplore.ieee.org/document/9002699/ (besucht am 12. 11. 2020).

[Fir21] Mozilla Firefox. *Readability.js.* 2021. URL: https://github.com/mozilla/readability (besucht am 20. 04. 2021).

[For+09] Marc Alier Forment u. a. *Interoperability for LMS: The Missing Piece to Become the Common Place for Elearning Innovation.* 2009.

[Fow14] Martin Fowler. *Microservices.* 2014. URL: https://martinfowler.com/articles/microservices.html (besucht am 02. 05. 2021).

[Fow18] Martin Fowler. *Refactoring: improving the design of existing code.* Addison-Wesley Professional, 2018.

[Fri05] Norm Friesen. „Interoperability and Learning Objects: An Overview of E-Learning Standardization“. In: *Interdisciplinary Journal of e-Skills and Lifelong Learning* 1 (2005), S. 023–031. URL: https://www.informingscience.org/Publications/408 (besucht am 04. 12. 2020).

[FSC20] Jibril Frej, Didier Schwab und Jean-Pierre Chevallet. *Knowledge Based Transformer Model for Information Retrieval.* 2020.

[Gal+13] Daniel Gallego u. a. „Enhanced recommendations for e-Learning authoring tools based on a proactive context-aware recommender“. In: *2013 IEEE Frontiers in Education Conference (FIE).* 2013 IEEE Frontiers in Education Conference (FIE). Oklahoma City, OK, USA: IEEE, Okt. 2013, S. 1393–1395. URL: http://ieeexplore.ieee.org/document/6685060/ (besucht am 12. 11. 2020).

[GBQ17] Aldo Gordillo, Enrique Barra und Juan Quemada. „An easy to use open source authoring tool to create effective and reusable learning objects: AN EASY TO USE LEARNING OBJECT AUTHORING TOOL“. In: *Computer Applications in Engineering Education* 25.2 (März 2017), S. 188–199. URL: http://doi.wiley.com/10.1002/cae.21789 (besucht am 12. 11. 2020).

[GCP03] Susan Gauch, Jason Chaffee und Alexander Pretschner. *Ontology-based personalized search and browsing.* 2003.

[Gho20] Pranab Ghosh. *Semantic Search with Pre Trained Neural Transformer Models using Document, Sentence and Token Level Embeddings.* 28. Juli 2020.

[Glo21] Articulate Global. *Articulate 360.* 2021. URL: https://360.articulate.com (besucht am 20. 04. 2021).

[GT15] C. Gormley und Zachary Tong. *Elasticsearch: The Definitive Guide.* 2015.

[GW09] Jürgen Grosche und Michael Wunder. „Grundlagen der Interoperabilität“. In: *Verteilte Führungsinformationssysteme.* Hrsg. von Michael Wunder und Jürgen Grosche. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, S. 19–26. URL: http://link.springer.com/10.1007/978-3-642-00509-1_2 (besucht am 04. 12. 2020).

[Her+06] William R Hersh u. a. „TREC 2006 genomics track overview.“ In: *TREC.* Bd. 7. 2006, S. 500–274.

[HI13] Haytham W. Hijazi und Jamil A. Itmazi. *Smart Crawler Based e-Learning.* 2013.

[HRR19] Inma Hernández, Carlos R. Rivero und David Ruiz. „Deep Web crawling: a survey“. In: *World Wide Web* 22.4 (Juli 2019), S. 1577–1610. URL: http://link.springer.com/10.1007/s11280-018-0602-1 (besucht am 13. 11. 2020).

[Ian19] Bogdan Iancu. „Web Crawler for Indexing Video e-Learning Resources: A YouTube Case Study“. In: *Informatica Economica* 23.2 (2019), S. 15–24.

[IEE02] IEEE. „IEEE Standard for Learning Object Metadata“. In: *IEEE Std 1484.12.1-2002* (2002), S. 1–40.

[IMS03] IMS. *IMS LIP Specification*. 2003.

[IMS15] IMS. *IMS Common Cartridge Specification*. 2015. URL: http://www.imsglobal.org/cc/index.html (besucht am 02. 05. 2021).

[IMS19a] IMS. *LTI 1.1 Implementation Guide*. 2019. URL: https://www.imsglobal.org/specs/ltiv1p1p1/implementation-guide (besucht am 02. 05. 2021).

[IMS19b] IMS. *LTI Advantage Implementation Guide*. 2019. URL: https://www.imsglobal.org/spec/lti/v1p3/impl (besucht am 02. 05. 2021).

[IMS20] IMS. *IMS QTI Specification*. 2020. URL: https://www.imsglobal.org/question/index.html (besucht am 06. 05. 2021).

[Inc21] edX Inc. *Open EdX Developer Guide*. 2021. URL: https://edx.readthedocs.io/projects/edx-developer-guide/en/latest/index.html (besucht am 05. 05. 2021).

[Ini17] ADL Initiative. *xAPI-Spec*. 2017. URL: https://github.com/adlnet/xAPI-Spec (besucht am 02. 05. 2021).

[IOS21] Fraunhofer IOSB. *Common Learning Middleware*. 2021. URL: http://s.fhg.de/fhgclm (besucht am 06. 05. 2021).

[IY19] Mohammed Ibrahim und Yanyan Yang. „An Ontology-based Web Crawling Approach for the Retrieval of Materials in the Educational Domain:“ In: *Proceedings of the 11th International Conference on Agents and Artificial Intelligence*. 11th International Conference on Agents and Artificial Intelligence. Prague, Czech Republic: SCITEPRESS - Science und Technology Publications, 2019, S. 900–906. URL: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0007692009000906 (besucht am 03. 12. 2020).

[JKN07] Mohamed Jemni, Mohamed Koutheaïr Khribi und Olfa Nasraoui. *Toward a Hybrid Recommender System for E-Learning Personalization Based on Web Usage Mining Techniques and Information Retrieval*. 2007.

[KHK15] Maryam Khademi, Maryam Haghshenas und Hoda Kabir. *A Review On Authoring Tools*. 2015.

[KIK13] Ville Karavirta, Petri Ihantola und Teemu Koskinen. „Service-Oriented Approach to Improve Interoperability of E-Learning Systems“. In: *2013 IEEE 13th International Conference on Advanced Learning Technologies*. 2013 IEEE 13th International Conference on Advanced Learning Technologies (ICALT). Beijing, China: IEEE, Juli 2013, S. 341–345. URL: http://ieeexplore.ieee.org/document/6601947/ (besucht am 23. 11. 2020).

[KJN08] Mohamed Koutheaïr Khribi, Mohamed Jemni und Olfa Nasraoui. „Automatic Recommendations for E-Learning Personalization Based on Web Usage Mining Techniques and Information Retrieval“. In: *2008 Eighth IEEE International Conference on Advanced Learning Technologies*. 2008 Eighth IEEE International Conference on Advanced Learning Technologies. Santander, Cantabria, Spain: IEEE, 2008, S. 241–245. URL: http://ieeexplore.ieee.org/document/4561676/ (besucht am 12. 11. 2020).

[KP11] Brijesh Kumar und Saurabh Pal. „Mining Educational Data to Analyze Students Performance“. In: *International Journal of Advanced Computer Science and Applications* 2.6 (2011). URL: http://thesai.org/Publications/ViewPaper?Volume=2&Issue=6&Code=IJACSA&SerialNo=9 (besucht am 03. 12. 2020).

[Lee+19] Jinhyuk Lee u. a. „BioBERT: a pre-trained biomedical language representation model for biomedical text mining“. In: *Bioinformatics* (Sep. 2019). Hrsg. von JonathanEditor Wren. URL: http://dx.doi.org/10.1093/bioinformatics/btz682.

[Leh+15] Jens Lehmann u. a. „DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia“. In: *Semantic Web* 6.2 (2015), S. 167–195. URL: https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/SW-140134 (besucht am 27. 03. 2021).

[Liu+19] Yinhan Liu u. a. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[Lop+20] Sonsoles Lopez-Pernas u. a. „Ediphy: A modular and extensible open-source web authoring tool for the creation of interactive learning resources“. In: *2020 16th International Conference on Intelligent Environments (IE)*. 2020 16th International Conference on Intelligent Environments (IE). Madrid, Spain: IEEE, Juli 2020, S. 115–121. URL: https://ieeexplore.ieee.org/document/9154949/ (besucht am 12. 11. 2020).

[Lu04] Jie Lu. *A Personalized e-Learning Material Recommender System*. 2004.

[LW71] Michael Levandowsky und David Winter. „Distance between sets". In: *Nature* 234.5323 (1971), S. 34–35.

[LZ10] Xinjin Li und Sujing Zhang. „Application of Web Usage Mining in e-learning Platform". In: *2010 International Conference on E-Business and E-Government*. 2010 International Conference on E-Business and E-Government (ICEE). Guangzhou, China: IEEE, Mai 2010, S. 1391–1394. URL: http://ieeexplore.ieee.org/document/5591773/ (besucht am 03. 12. 2020).

[Mar02] Robert C Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[Men+11] Pablo N. Mendes u. a. „DBpedia spotlight: shedding light on the web of documents". In: *Proceedings of the 7th International Conference on Semantic Systems - I-Semantics '11*. the 7th International Conference. Graz, Austria: ACM Press, 2011, S. 1–8. URL: http://dl.acm.org/citation.cfm?doid=2063518.2063519 (besucht am 10. 04. 2021).

[Moo21a] Moodle. *Moodle Demo*. 2021. URL: https://school.moodledemo.net/ (besucht am 05. 05. 2021).

[Moo21b] Moodle. *Moodle Developer Documentation*. 2021. URL: https://docs.moodle.org/dev/Main_Page (besucht am 05. 05. 2021).

[Mor17] Marco Morales. *Update: Course Navigation Changes*. 2017. URL: https://open.edx.org/announcements/update-course-navigation-changes/ (besucht am 05. 05. 2021).

[MRS08] Christopher D. Manning, Prabhakar Raghavan und Hinrich Schütze. *Introduction to information retrieval*. OCLC: ocn190786122. New York: Cambridge University Press, 2008. 482 S.

[NH02] Marc Najork und Allan Heydon. „High-Performance Web Crawling". In: *Handbook of Massive Data Sets*. Hrsg. von James Abello, Panos M. Pardalos und Mauricio G. C. Resende. Bd. 4. Series Title: Massive Computing. Boston, MA: Springer US, 2002, S. 25–45. URL: http://link.springer.com/10.1007/978-1-4615-0005-6_2 (besucht am 13. 11. 2020).

[Nie08] Helmut M. Niegemann. *E-Learning-Standards und Standardisierung*. 2008.

[NOC11] Prakash M Nadkarni, Lucila Ohno-Machado und Wendy W Chapman. „Natural language processing: an introduction“. In: *Journal of the American Medical Informatics Association* 18.5 (2011), S. 544–551.

[NZ10] Olfa Nasraoui und Leyla Zhuhadar. *Improving Recall and Precision of a Personalized Semantic Search Engine for E-learning*. 2010.

[OCC13] Jonathan Oliver, Chun Cheng und Yanggui Chen. „TLSH – A Locality Sensitive Hash“. In: *2013 Fourth Cybercrime and Trustworthy Computing Workshop*. 2013 Fourth Cybercrime and Trustworthy Computing Workshop (CTC). Sydney NSW, Australia: IEEE, Nov. 2013, S. 7–13. URL: http://ieeexplore.ieee.org/document/6754635/ (besucht am 02. 04. 2021).

[ON10] Christopher Olston und Marc Najork. „Web Crawling“. In: *Foundations and Trends® in Information Retrieval* 4.3 (2010), S. 175–246. URL: http://www.nowpublishers.com/article/Details/INR-017 (besucht am 13. 11. 2020).

[PG11] K R Premlatha und T V Geetha. *Focused Crawling for Educational Materials from the Web*. 2011.

[PO05] Sandeep Pandey und Christopher Olston. „User-centric Web crawling“. In: *Proceedings of the 14th international conference on World Wide Web - WWW '05*. the 14th international conference. Chiba, Japan: ACM Press, 2005, S. 401. URL: http://portal.acm.org/citation.cfm?doid=1060745.1060805 (besucht am 27. 11. 2020).

[PR04] Jinsoo Park und Sudha Ram. „Information Systems Interoperability: What Lies Beneath?“ In: *ACM Transactions on Information Systems* 22.4 (2004), S. 38.

[PS17] Kolli Pavani und G P Sajeev. *A Novel Web Crawling Method For Vertical Search Engines*. 2017.

[PSM14] Jeffrey Pennington, Richard Socher und Christopher D Manning. „Glove: Global vectors for word representation“. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, S. 1532–1543.

[RA18] Mohammad Mustaneer Rahman und Nor Aniza Abdullah. „A Personalized Group-Based Recommendation Approach for Web Search in E-Learning“. In: *IEEE Access* 6 (2018), S. 34166–34178. URL: https://ieeexplore.ieee.org/document/8395492/ (besucht am 25. 04. 2021).

[Ram+03] Juan Ramos u. a. „Using tf-idf to determine word relevance in document queries“. In: *Proceedings of the first instructional conference on machine learning*. Bd. 242. 1. Citeseer. 2003, S. 29–48.

[RCV15] André Luís Alice Raabe, Agnaldo Costa und Marli Fátima Vick Vieira. *Development and Evaluation of an Authoring Tool Taxonomy*. 2015.

[Reu+16] Ralf H Reussner u. a. *Modeling and simulating software architectures: The Palladio approach*. MIT Press, 2016.

[RG19] Nils Reimers und Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 27. Aug. 2019.

[Rus18] Alexander Rush. „The Annotated Transformer“. In: *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*. Melbourne, Australia: Association for Computational Linguistics, Juli 2018, S. 52–60. URL: https://www.aclweb.org/anthology/W18-2509.

[Sch+07] J Ben Schafer u. a. „9 Collaborative Filtering Recommender Systems“. In: *The Adative Web* (2007), S. 34.

[SV88] Alan F Smeaton und CJ Van Rijsbergen. „Experiments on incorporating syntactic processing of user queries into a document retrieval strategy“. In: *Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval*. 1988, S. 31–51.

[TGS05] D Taibi, M Gentile und L Seta. „A Semantic Search Engine for Learning Resources“. In: *Recent Research Developments in Learning Technologies* (2005).

[The+16] Matthias Then u. a. „Innovative Authoring Tools for Online-Courses with Assignments - Integrating Heterogeneous Tools of e-Learning Platforms into Hybrid Application Solutions“. In: *International Journal of Emerging Technologies in Learning (iJET)* 11.2 (23. Feb. 2016), S. 12.

[TM03] Tiffany Ya Tang und Gordon Mccalla. *Smart Recommendation for an Evolving E-Learning System*. 2003.

[TNM18] John K. Tarus, Zhendong Niu und Ghulam Mustafa. „Knowledge-based recommendation: a review of ontology-based recommender systems for e-learning“. In: *Artificial Intelligence Review* 50.1 (Juni 2018), S. 21–48.

[TNY17] John K. Tarus, Zhendong Niu und Abdallah Yousif. *A hybrid knowledge-based recommender system for e-learning based on ontology and sequential pattern mining*. 2017.

[UMN07] K Umadevi, B Uma Maheswari und P Nithya. *Design of E-Learning Application through Web Mining*. 2007.

[Vas+17] Ashish Vaswani u. a. „Attention Is All You Need“. In: *arXiv:1706.03762 [cs]* (5. Dez. 2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (besucht am 18. 04. 2021).

[Wol+20] Thomas Wolf u. a. „Transformers: State-of-the-Art Natural Language Processing“. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Okt. 2020, S. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[WW03] Hartmut Wandke und Elke Wetzenstein-Ollenschläger. *Assistenzsysteme: Woher und Wohin?* 2003.

[Zai02] O.R. Zaiane. „Building a recommender agent for e-learning systems“. In: *International Conference on Computers in Education, 2002. Proceedings.* International Conference on Computers in Education. Bd. 1. Auckland, New Zealand: IEEE Comput. Soc, 2002, S. 55–59. URL: http://ieeexplore.ieee.org/document/1185862/ (besucht am 12. 11. 2020).

[Zha+16] Feng Zhao u. a. „SmartCrawler: A Two-Stage Crawler for Efficiently Harvesting Deep-Web Interfaces“. In: *IEEE Transactions on Services Computing* 9.4 (1. Juli 2016), S. 608–620. URL: http://ieeexplore.ieee.org/document/7064719/ (besucht am 13. 11. 2020).

Tabellenverzeichnis

Abbildungsverzeichnis

Glossar

ADL Die Advanced Distributed Learning Initiative (ADL) ist eine Organisation, die sich mit der Entwicklung von E-Learning-Standards beschäftigt, insb. SCORM [ADL09] und xAPI [Ini17]. 4, 11

CLM Die Common Learning Middleware (CLM) ist eine am Fraunhofer IOSB entwickelte Middleware für die Verbindung von LMS mit E-Learning-Standards [IOS21]. 1, 2, 4, 34, 40, 51, 52, 54, 74–76, 84, 85, 87–90, 92, 93, 111, 113

Common Cartridge Common Cartrige (CC) ist ein Standard für die Spezifikation und Strukturierung von E-Learning-Ressourcen, um sie zwischen Systemen zu transferieren [IMS15]. 2, 10, 12, 52, 58, 74, 92

Deep Link Eine URI, die auf einen Teil des Deep Web verweist, also auf Inhalte, die nicht ohne weitere Technologien direkt aufgerufen bzw. gefunden werden können. 1, 2, 40, 54, 79

IMS Das IMS Global Consortium [Con21] ist eine Organisation, die sich mit der Entwicklung von E-Learning-Standards beschäftigt, insb. LTI [IMS19a] und QTI [IMS20]. 1, 4, 10, 12, 32, 52

LIP IMS Learner Information Package [IMS03] ist ein Standard für die Beschreibung von Nutzer-Informationen im E-Learning-Kontext.. 32

LMS Ein Learning Management System (LMS) ist eine Anwendung, die die Erstellung, Verwaltung und Präsentation von E-Learning-Inhalten erlaubt. Die hier diskutierten Beispiele für solche Anwendungen sind Moodle, ILIAS und open edX. 1, 2, 4, 9, 11, 13, 19, 20, 23–25, 34, 51, 52, 54–59, 75, 79, 80, 85, 87, 90, 92, 93

LOM Learning Object Metadata (LOM) [IEE02] ist ein IEE-Standard für die Beschreibung von Metadaten von E-Learning-Objekten.. 12, 14, 20, 32

LRS Ein Learning Record Store (LRS) ist eine Komponente, die für die Datenhaltung von E-Learning-Ergebnissen verantwortlich ist. 12, 34

LTI Learning Tools Interoperability (LTI) ist ein Standard, um E-Learning-Inhalte eines Systems in ein anderes einzubinden. Benutzt werden in der Regel LTI 1.1 [IMS19a] oder LTI 1.3 [IMS19b]. 1, 2, 4, 12, 13, 24, 25, 28, 32–36, 40, 45, 54, 56, 58, 59, 74, 75, 79, 84, 86–90, 111, 113

NLP Natural Language Processing (NLP) ist eine Forschungsrichtung der Informatik, die sich mit der Verarbeitung und dem Verständnis der natürlichen Sprache beschäftigt. 41, 46, 47, 53, 62, 65, 82

QTI Question Tool Interoperability (QTI) ist ein Standard für den Export bzw. Import von Quizzen bzw. Fragen im E-Learning-Kontext. 4, 12, 32, 34

REST-API Ein Representational State Transfer (REST) Application Programming Interface (API) ist der de-facto Standard für die Implementierung von HTTP-Schnittstellen zwischen Systemen.. 13, 34, 73

SCORM Das Shareable Content Object Reference Model (SCORM) ist ein Standard für den Export bzw. Import von E-Learning-Inhalten. 7, 9, 11, 13, 32

xAPI Die Experience API (xAPI) ist ein von der ADL entwickelter Standard für das Speichern von E-Learning-Ergebnissen. 4, 7, 12, 34

xBlock xBlocks sind eine Technologie des LMS Open edX, die eine direkte Adressierung von Inhalten ermöglicht. 13, 28, 56

Die Klassen des Systems in der Übersicht

Klasse	Beschreibung
App.component	Basis der Applikation
Landing.component	Landing Page, erscheint nach Öffnen der Applikation bzw. nach LTI-Launch
Navigation.component	Implementiert die Nav-Bar inkl. des dortigen Dropdown
Editor.component	Enthält den Text-Editor, den semantischen Editor in Baum-Form und ein Menü
Course.component	Rekursiv aufgerufene Komponenten des Kurs-Baums, sie repräsentieren jeweils eine Node
Search.component	Komponente, die parallel zur Editor-Komponente angezeigt wird, sie enthält die Search-Box und den Container für die Ergebnisse sowie einen Tab für Content-Vorschläge
Search-Settings.component	Kapselung der aktuellen Sucheinstellungen, insb. den Gewichten für die semantische Suche.
Results.component	Stellt Suchergebnisse dar
Visibility.service	Kontrolliert zentral welche Teile des Kursbaum sichtbar sind und bei re-render auch so bleiben
DragDrop.service	Wrapper um CDK.DragDrop, fügt Unterstützung für geschachtelte Listen hinzu
Search.service	Für Kommunikation mit Findoo.Server/*, aufgerufen im Rahmen von Suchen und Autocomplete
Recommender.service	Kommunikation mit Findoo.Server/recommend, aufgerufen wenn neuer Inhalt hinzugefügt
Launch.service	Kommunikation mit Findoo.Server/launch oder, abhängig vom Betriebsmodus, mit dem einbettenden Window-Objekt

Tabelle 1: Die Klassen des Frontends

Klasse	Beschreibung
Embeddings	Microservice, der die semantischen Einbettungen berechnet
EmbeddingAdapter	Adapterklasse für die Kommunikation des Systems mit dem Embeddings-Microservice
Mock-CLM	Microservice, der die für den lokalen Betriebsmodus benötigte CLM-Funktionalität implementiert
LMSCrawler	Basisklasse für alle Crawler, hier ist alle nötige Logik implementiert
MoodleCrawler	Subklasse von LMSCrawler, passt Attribute und Methoden an Moodle an
IliasCrawler	Subklasse von LMSCrawler, passt Attribute und Methoden an Ilias an
EDXCrawler	Subklasse von LMSCrawler, passt Attribute und Methoden an Open edX an
SimpleCrawler	Subklasse von LMSCrawler, diese indiziert eine beliebige Website, traversiert diese aber nicht
LMSClassifier	Bestimmen des LMS-Typs einer gegebenen Website
PageClassifier	Klassifikation des Inhaltstyp einer gegebenen Website anhand einer Menge von Ankerpunkten
DuplicateDetection	TLSH-basierte Near Duplicate Detection
FindBodyHeuristic	Bestimmung des natürlichsprachigen Bodies einer Website
LanguageDetection	Bestimmung der Sprache, in der ein gegebener Inhalt geschrieben wurde
TextExtraction	Extraktion von Text aus PDF-Dateien
IndexOutput	Adapterklasse für die Kommunikation des Crawlers mit dem Index
IndexInput	Kapselung der Input-Logik für den Index
QueryDecorator	Erstellung von Anfragen an den Index in dessen proprietären Sprache
SearchHandler	Kapselung der Such-Logik
CLMAdapter	Adapterklasse für die Kommunikation mit der CLM
OntologyAdapter	Adapterklasse für die Kommunikation des Systems mit einem Ontologie-Modul
RecommenderAdapter	Adapterklasse für die Kommunikation des Systems mit einem Recommender-Modul
index	Die Server-Instanz, Implementierung der Endpoints der API

Tabelle 2: Die Klassen des Backends

Ergänzende Diagramme

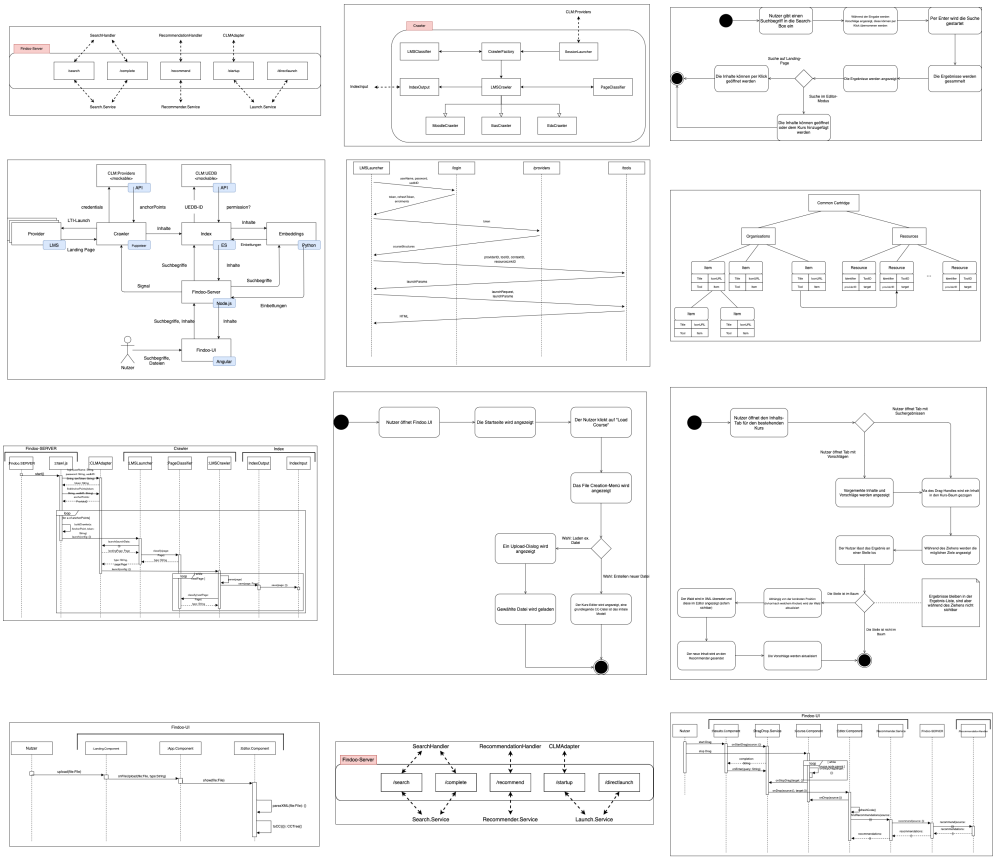


Abbildung 1: Die ergänzenden Diagramme.

In dem mit dieser Arbeit assoziierten Gitlab-Repository[1] sind die in Abbildung 1 dargestellten UML Sequenz- und Aktivitätsdiagramme sowie Architekturdiagramme verfügbar.

1 https://gitlab-ext.iosb.fraunhofer.de/clm/findoo/mscschwarz2020/thesis