Fachbereich VI - Informatik und Medien

Masterarbeit

im Studiengang Medieninformatik

zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

Thema: Evaluation und Optimierung einer routefähigen Middleware-Architektur im Bereich E-Learning

Autor: François Dubois
MatNr. 806958

Version vom: 27. Februar 2017

1. Betreuerin: Prof. Dr. Merceron
Gutachter: Prof. Dr. Konert
Kurzzusammenfassung

In dieser Arbeit wird eine Middleware Infrastructure (MI) vorgestellt, die in erster Linie die interoperable Nutzung verschiedener Learning-Management-Systeme (LMS) über eine zentrale REST-API ermöglichen soll. Das wesentliche Merkmal der MI ist ein Konzept, bei dem nicht nur unterschiedliche LMS unterstützt werden, sondern auch dynamisch mehrere Server über sogenannte Routen angesprochen werden können. Dies ermöglicht, dass eine Clientanwendung in der E-Learning-Branche verschiedene LMS an separaten Bildungseinrichtungen anbieten kann. Die Art der Implementierung soll dieselbe Komplexität sowie dieselben Merkmale aufweisen, wie die Integration für nur eine Bildungseinrichtung. Ein langfristiges Ziel ist es, die REST-APIs in Form von Microservices auch in anderen Branchen einsetzen zu können. Im Vergleich zu verwandten serviceorientierten Architekturen wird bei diesem Ansatz eine direkte Kommunikation zwischen Service Providern, in diesem Fall Bildungseinrichtungen, vermieden.

Abstract

This paper presents a middleware infrastructure (MI) whose primary purpose is enabling the usage of various learning management systems (LMS) via a central REST-API. The MI’s core feature is a concept that not only supports several LMS but also allows to dynamically query multiple servers via routes. This allows a client application for e-learning purposes to offer several LMS to separate educational institutions. This type of implementation should possess the same complexity and attributes as the integration for a single institution. The long-term goal is applying the REST-APIs in other business sectors in the form of microservices. Compared to other service-oriented structures, this approach does not necessitate direct communication between different service providers, in this case educational institutions.
## Inhaltsverzeichnis

<table>
<thead>
<tr>
<th>Abteilung</th>
<th>Seiten</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abbildungsverzeichnis</td>
<td>5</td>
</tr>
<tr>
<td>Tabellenverzeichnis</td>
<td>6</td>
</tr>
<tr>
<td>Listingverzeichnis</td>
<td>7</td>
</tr>
<tr>
<td>Abkürzungsverzeichnis</td>
<td>8</td>
</tr>
<tr>
<td>Glossar</td>
<td>9</td>
</tr>
</tbody>
</table>

### 1 Einleitung
1.1 Motivation ................................................. 13
1.2 Problemstellung ........................................... 14
1.3 Zielsetzung ................................................. 14
1.4 Methodik .................................................. 15
1.5 Aufbau der Arbeit ......................................... 16

### 2 Related Work
2.1 Die Bedeutung von Learning-Management-Systemen (LMS) ........ 17
2.2 Die heterogene Systemlandschaft von LMS ..................... 17
2.3 Eine historische Betrachtung von LMS ........................ 19
2.4 Das Potenzial von serviceorientierten Ansätzen in der E-Learning-Branche ........................................... 21
2.5 Architekturkonzepte von serviceorientierten Ansätzen in der E-Learning-Branche ........................................... 23
2.6 Zusammenfassung ............................................. 31

### 3 Systemanalyse
3.1 Das zu lösende Problem und der Mehrwert des Systems ........ 32
3.2 Ein branchenübergreifender Produkteinsatz .................... 33
3.3 Die Anforderungsspezifikationen als Grundlage für den Systementwurf ........................................... 36
  3.3.1 Muss-Kriterien ........................................... 36
  3.3.2 Soll-Kriterien ........................................... 37
  3.3.3 Kann-Kriterien ........................................... 37
  3.3.4 Abgrenzungskriterien .................................... 37
3.4 Zusammenfassung ............................................. 38

### 4 Systementwurf
4.1 Die Middleware Infrastructure (MI) .......................... 39
4.2 Der Hub ....................................................... 40
  4.2.1 Die Kommunikation zwischen Hub und Middleware API .... 42
  4.2.2 Der Aufbau und die Eigenschaften einer Route ............. 43
  4.2.3 Das Hub-Management-Interface ............................ 44
4.3 Die Middleware API (MWA) .................................... 45
  4.3.1 Dynamisierung von Verbindungswegen (Routing) .......... 45
  4.3.2 Die Drittanbieter-Authentifizierung von User und Consumer ... 47
  4.3.3 Das Individualisieren von Routen ......................... 48
  4.3.4 Die Bypass-Authentifizierung (User) ....................... 48
  4.3.5 Das Aufsuchen und Beschreiben von Third Party Modules ... 49
4.4 Der DAO Manager als Service Broker .......................... 51
4.5 Skalierung und Load Balancing ................................ 52
4.6 Die Integration von Drittanbietern aus der E-Learning-Branche .......................... 53
4.7 Zusammenfassung .......................................................... 55

5 Implementierung .......................................................... 56
5.1 Die logische Codestruktur der Middleware Infrastructure ......................... 56
  5.1.1 Die Bereitstellung von Data Access Objects (DAO) ..................... 57
  5.1.2 Die Bereitstellung von Data Transfer Objects (DTO) .................. 62
  5.1.3 Die Fehlerbehandlung der Middleware Infrastructure (Exception) ...... 63
  5.1.4 Die Integration von DAO-Verbindungstypen (Connector) ............ 63
  5.1.5 Die Konfiguration von Hub und Middleware API (Config) .......... 64
  5.1.6 Das Management von Routen (Routing) ............................. 64
  5.1.7 Das Abfragen von verfügbaren Modulen (Scan) ....................... 66
  5.1.8 Die modulübergreifenden Funktionalitäten (Shared) ................ 68
  5.1.9 Die Implementierung von Middleware APIs ........................... 68
5.2 Die Schnittstellen der Hub API ......................................... 73
5.3 Zusammenfassung .......................................................... 74

6 Evaluation .............................................................. 76
6.1 Qualitative Evaluation .................................................... 76
  6.1.1 Anwendungsfälle (Kurse) ........................................... 76
  6.1.2 Die Produktumgebung (Integration) ................................ 77
  6.1.3 Bewertung der Kernziele ............................................ 78
6.2 Quantitative Evaluation ................................................... 81
  6.2.1 Die Testumgebung .................................................. 81
  6.2.2 Testszenarien zur Ermittlung der Performance von MWAs .......... 82
  6.2.3 Ergebnisse der Testszenarien .................................... 83
6.3 Zusammenfassung .......................................................... 87

7 Zusammenfassung .......................................................... 89
7.1 Fazit ............................................................................ 89
7.2 Ausblick ....................................................................... 92
7.3 Schlusswort .................................................................. 94

Literaturverzeichnis ........................................................... 95

Anhang .................................................................

A LRS Statement .......................................................... 99
B LMS Course .............................................................. 100
C Hub Management .......................................................... 101

Eidesstattliche Erklärung ................................................. 102
Abbildungsverzeichnis

<table>
<thead>
<tr>
<th>Abbildungsnummer</th>
<th>Bildschlagzeile</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Share of LMS market worldwide in 2013, by vendor</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>Modular deployment</td>
<td>22</td>
</tr>
<tr>
<td>3</td>
<td>find-bind-execute paradigm</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>Generations of learning management systems</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>The layered model</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>E-Learning Grid</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>Service architecture of e-learning system</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>Joint Degrees in E-Learning Systems</td>
<td>28</td>
</tr>
<tr>
<td>9</td>
<td>Dynamic personalized E-Learning scenario</td>
<td>29</td>
</tr>
<tr>
<td>10</td>
<td>Open edX MOOC Architecture</td>
<td>30</td>
</tr>
<tr>
<td>11</td>
<td>Abstract Middleware Infrastructure (MI)</td>
<td>33</td>
</tr>
<tr>
<td>12</td>
<td>Middleware Infrastructure (MI)</td>
<td>39</td>
</tr>
<tr>
<td>13</td>
<td>Hub and Spoke</td>
<td>41</td>
</tr>
<tr>
<td>14</td>
<td>Enterprise Service Bus</td>
<td>41</td>
</tr>
<tr>
<td>15</td>
<td>Hub Communication</td>
<td>42</td>
</tr>
<tr>
<td>16</td>
<td>Route</td>
<td>43</td>
</tr>
<tr>
<td>17</td>
<td>Showcase Hub Management</td>
<td>45</td>
</tr>
<tr>
<td>18</td>
<td>Middleware Routing</td>
<td>46</td>
</tr>
<tr>
<td>19</td>
<td>Consumer and User Authentication</td>
<td>47</td>
</tr>
<tr>
<td>20</td>
<td>Individualized Routes</td>
<td>48</td>
</tr>
<tr>
<td>21</td>
<td>Bypass Authentication</td>
<td>49</td>
</tr>
<tr>
<td>22</td>
<td>Route Domain (Type)</td>
<td>50</td>
</tr>
<tr>
<td>23</td>
<td>Middleware Balancing</td>
<td>53</td>
</tr>
<tr>
<td>24</td>
<td>Middleware Infrastructure (E-Learning)</td>
<td>54</td>
</tr>
<tr>
<td>25</td>
<td>Module integration dependencies</td>
<td>57</td>
</tr>
<tr>
<td>26</td>
<td>Class diagram Forum</td>
<td>69</td>
</tr>
<tr>
<td>27</td>
<td>xAPI Statement structure</td>
<td>71</td>
</tr>
<tr>
<td>28</td>
<td>Parts of IMS CC specification</td>
<td>72</td>
</tr>
<tr>
<td>29</td>
<td>SLHw openLCMSI</td>
<td>78</td>
</tr>
<tr>
<td>30</td>
<td>Test Environment (Middleware API)</td>
<td>81</td>
</tr>
<tr>
<td>31</td>
<td>Performance in Bytes - Middleware APIs vs. Third Parties</td>
<td>86</td>
</tr>
<tr>
<td>32</td>
<td>Marktanteile Hersteller Wearables 2014/2015</td>
<td>93</td>
</tr>
<tr>
<td>33</td>
<td>LRS Statement</td>
<td>99</td>
</tr>
<tr>
<td>34</td>
<td>LMS Course</td>
<td>100</td>
</tr>
<tr>
<td>35</td>
<td>Edit Middleware Consumer</td>
<td>101</td>
</tr>
</tbody>
</table>
Tabellenverzeichnis

1 E-Learning Standards .................................................. 18
2 Key Figures of Selected MOOC Platforms ............................ 21
3 IAF Application Services .................................................. 25
4 Mapping between Third Party and Middleware API ................. 35
5 REST Mapping .............................................................. 51
6 API Standards ............................................................... 53
7 Hub API mappings .......................................................... 73
8 Functional modules ......................................................... 74
9 Hardware specifications - Server ....................................... 81
10 Scenarios 1 and 2 with 10.000 Requests - 100 Threads .......... 84
11 Scenarios 1 and 2 with 25.000 Requests - 250 Threads .......... 84
12 Scenarios 3 and 4 with 250 Requests - 50 Threads ............... 85
13 Performance Middleware APIs vs. Third Parties ................. 86
14 Performance factors ...................................................... 87
## Listingverzeichnis

<table>
<thead>
<tr>
<th></th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CRUD Methods - AbstractRouteDAO</td>
<td>58</td>
</tr>
<tr>
<td>2</td>
<td>DAO lookup method - DAOManager</td>
<td>59</td>
</tr>
<tr>
<td>3</td>
<td>Dynamic JNDI</td>
<td>59</td>
</tr>
<tr>
<td>4</td>
<td>Find CourseDAO</td>
<td>60</td>
</tr>
<tr>
<td>5</td>
<td>Filter and DAO registration</td>
<td>61</td>
</tr>
<tr>
<td>6</td>
<td>ContainerResponse - JSON</td>
<td>62</td>
</tr>
<tr>
<td>7</td>
<td>ContainerResponse - Builder Pattern</td>
<td>62</td>
</tr>
<tr>
<td>8</td>
<td>Property injection</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>Add Route to HubStorage</td>
<td>65</td>
</tr>
<tr>
<td>10</td>
<td>Third Party Module (TPM) scan</td>
<td>66</td>
</tr>
<tr>
<td>11</td>
<td>Virtual file to physical file</td>
<td>67</td>
</tr>
<tr>
<td>Abkürzung</td>
<td>Definition/Erklärung</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>ACP</td>
<td>Adaptive Contextual Portal</td>
<td></td>
</tr>
<tr>
<td>ADL</td>
<td>Advanced Distributed Learning</td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>Consumer Credentials</td>
<td></td>
</tr>
<tr>
<td>CDI</td>
<td>Context and Dependency Injection</td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>Communication</td>
<td></td>
</tr>
<tr>
<td>CRM</td>
<td>Customer-Relationship-Management</td>
<td></td>
</tr>
<tr>
<td>CRUD</td>
<td>Create-, Read-, Update- und Delete-Methoden</td>
<td></td>
</tr>
<tr>
<td>DAO</td>
<td>Data Access Object</td>
<td></td>
</tr>
<tr>
<td>DTO</td>
<td>Data Transfer Object</td>
<td></td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBeans</td>
<td></td>
</tr>
<tr>
<td>ELF</td>
<td>The E-Learning Framework</td>
<td></td>
</tr>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
<td></td>
</tr>
<tr>
<td>FUMES</td>
<td>Federated User Modeling Exchange Service</td>
<td></td>
</tr>
<tr>
<td>GGF</td>
<td>Global Grid Forum</td>
<td></td>
</tr>
<tr>
<td>HR</td>
<td>Human Resource</td>
<td></td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
<td></td>
</tr>
<tr>
<td>IAF</td>
<td>IMS Abstract Framework</td>
<td></td>
</tr>
<tr>
<td>IDA</td>
<td>Independently Deployed Application</td>
<td></td>
</tr>
<tr>
<td>IMS CC</td>
<td>IMS Common Cartridge</td>
<td></td>
</tr>
<tr>
<td>IMS LTI</td>
<td>IMS Learning Tools Interoperability</td>
<td></td>
</tr>
<tr>
<td>IMS QTI</td>
<td>IMS Question &amp; Test Interoperability</td>
<td></td>
</tr>
<tr>
<td>IoC</td>
<td>Inversion of Control</td>
<td></td>
</tr>
<tr>
<td>IRI</td>
<td>Internationalized Resource Identifier</td>
<td></td>
</tr>
<tr>
<td>JNDI</td>
<td>Java Naming and Directory Interface</td>
<td></td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>Learning Analytics</td>
<td></td>
</tr>
<tr>
<td>LCMS</td>
<td>Learning Content Management System</td>
<td></td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
<td></td>
</tr>
<tr>
<td>LMS</td>
<td>Learning Management System</td>
<td></td>
</tr>
<tr>
<td>LRS</td>
<td>Learning Record Store</td>
<td></td>
</tr>
<tr>
<td>LTI</td>
<td>Learning Tool Interoperability</td>
<td></td>
</tr>
<tr>
<td>MI</td>
<td>Middleware Infrastructure</td>
<td></td>
</tr>
<tr>
<td>MIP</td>
<td>Middleware Infrastructure Provider</td>
<td></td>
</tr>
<tr>
<td>MOOC</td>
<td>Massive Open Online Course</td>
<td></td>
</tr>
<tr>
<td>MWA</td>
<td>Middleware API</td>
<td></td>
</tr>
<tr>
<td>OCCS</td>
<td>Open Corpus Content Service</td>
<td></td>
</tr>
<tr>
<td>OGSA</td>
<td>Open Grid Service Architecture</td>
<td></td>
</tr>
<tr>
<td>OSS</td>
<td>Open-Source-Software</td>
<td></td>
</tr>
<tr>
<td>PLE</td>
<td>Personal Learning Environment</td>
<td></td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
<td></td>
</tr>
<tr>
<td>RLO</td>
<td>Reusable Learning Object</td>
<td></td>
</tr>
<tr>
<td>SCORM</td>
<td>Sharable Content Object Reference Model</td>
<td></td>
</tr>
<tr>
<td>SOA</td>
<td>Serviceorientierte Architektur</td>
<td></td>
</tr>
<tr>
<td>TKB</td>
<td>Technical Knowledge Database</td>
<td></td>
</tr>
<tr>
<td>TPM</td>
<td>Third Party Module</td>
<td></td>
</tr>
<tr>
<td>UC</td>
<td>User Credentials</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
<td></td>
</tr>
<tr>
<td>USR</td>
<td>User</td>
<td></td>
</tr>
<tr>
<td>VF</td>
<td>Virtual File</td>
<td></td>
</tr>
<tr>
<td>VFS</td>
<td>Virtual File System</td>
<td></td>
</tr>
<tr>
<td>VO</td>
<td>Virtuelle Organisation</td>
<td></td>
</tr>
<tr>
<td>WAR</td>
<td>Web Application Archive</td>
<td></td>
</tr>
<tr>
<td>WS</td>
<td>Web-Service</td>
<td></td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
<td></td>
</tr>
</tbody>
</table>
Glossar

**ADLNet Vocabulary** Datenbank mit verschiedenen Definitionen für Verben, die über eine IRI in ExperienceAPI Statements verwendet werden können.

**Aggregation** Assoziation zwischen anwendungsspezifischen Daten.

**AngularJS** ein JavaScript-Framework von Google, wird zur Entwicklung von Webanwendungen verwendet.

**API Key** Zugangsschlüssel einer Middleware API für den Zugriff auf einen Hub.

**Application Programming Interface** Schnittstellen eines Systems, die anderen Anwendungen zur Verfügung stehen.

**Application Service** eine anwendungsspezifische Dienstleistung.

**Balancer Cluster** Gruppe von Systemen zur Lastenverteilung in einem verteilten Netzwerk.

**Balancer Member** System, das Clientanfragen bei einer Lastenverteilung annimmt.

**BASIC** Authentifizierungsmethode zum Übertragen von Username und Passwort über eine HTTP-Anfrage.

**BEARER** Authentifizierungsmethode zum Übertragen eines Tokens, der eine Zugangsberechtigung repräsentativ an den Besitzer dieses Tokens übergibt.

**Boilerplate Code** Codefragmente, die an verschiedenen Stellen in nahezu unveränderter Form vorliegen.

**Bottleneck** Der Begriff bezieht sich auf einen Engpass bei der Datenverarbeitung.

**Business Delegate** JavaEE-Entwurfsmuster, das die Präsentationsschicht von der Geschäftslogik entkoppelt.

**Clientanwendung** Anwendung, die von einem Endnutzer verwendet wird.

**Co-Design** beschreibt eine Synergie zwischen der digitalen Infrastruktur des SLHw-Projekts zur Bereitstellung von Lerninhalten und dem Analysieren von Lernaktivitäten (Learning Analytics).

**Common Service** eine Dienstleistung, die nicht anwendungsspezifisch und unter mehreren Anwendungen verbreitet ist, wie beispielsweise ein Authentifizierungsservice.

**Connector** Komponente der Middleware Infrastructure, die eine Verbindung zu einem Endpunkt herstellt und die Clientlogik für einen entsprechenden Service Provider besitzt.

**Consumer Credentials** Zugangsdaten für die administrative Nutzung von Dienstleistungen einer Drittanbieteranwendung.

**Consumer Token** Token für den Zugriff auf eine Middleware API.

**Context and Dependency Injection** beschreibt einen Java-Standard, indem Modulabhängigkeiten indirekt durch Dependency Injection realisiert werden.

**Corporate Design** Erscheinungsbild eines Unternehmens und im Rahmen dieser Arbeit die farbliche Gestaltung des Hub Management Interface.

**Customer-Relationship-Management** Pflege von Kundenbeziehungen in einem Unternehmen.

**DAO Manager** Der DAO Manager sucht und initialisiert unter Verwendung des JavaEE-Entwurfsmusters Business Delegate ein geeignetes Data Access Object.

**Data Access Object** Entwurfsmuster, das über ein Objekt den Zugriff auf eine Datenhaltungsschicht ermöglicht.

**Data Transfer Object** Entwurfsmuster, bei dem Daten in einem Objekt gebündelt über das Netzwerk übertragen werden.

**Dependency Injection** Benötigt ein Objekt eine Abhängigkeit zu einem anderen Objekt, wird dieses von einer zentralen Stelle angefordert und dem Objekt übergeben. Das Übergeben des Objektes wird als Dependency Injection bezeichnet.

**Deployment** Auslieferung (Distribution) einer Software an den Kunden.

**Domäne** Eine Domäne bezeichnet eine inhaltliche Spezialisierung und ist im Rahmen dieser Arbeit auf Branchen oder unabhängige Dienstleistungen zu beziehen.

**Drittanbieteranwendung** beliebige Anwendung, die Dienstleistungen bereitstellt, die vereinheitlicht werden sollen.

**E-Learning** Lernen über digitale Medien.

**Endpunkt** beschreibt den Zugriff auf einen Server mit einer Drittanbieteranwendung.

**Enterprise JavaBean** standardisierte Komponenten einer Java-EE Umgebung zur Entwicklung von mehrschichtigen Architekturen.

**Enterprise Service Bus** Netzwerkarchitektur, die den Datenaustausch zwischen Service Consumers und Service Providers über einen Bus realisiert.

**Entity** beschreibt im Rahmen dieser Arbeit das Datenobjekt, welches innerhalb des HTTP-Protokolls als Body übertragen wird und bei dem Representational State Transfer eine Ressource abbildet.
Experience API Standard zum Aufzeichnen von Lernaktivitäten, auch TinCan API genannt.

find-bind-execute-Paradigma beschreibt ein Paradigma, nach dem serviceorientierte Architekturen vorgehen, um das Auffinden, Integrieren und Nutzen von Dienstleistungen zu ermöglichen.

Geschäftslogik beschreibt in einer mehrschichtigen Architektur die Logik der Mittelschicht zwischen Datenerhaltungs- und Präsentationsschicht.

Grid Computing gemeinsame Nutzung von Ressourcen oder Rechenleistung in einem verteilten System. Im Vergleich zu einem Cluster sind die Systeme heterogen und lose gekoppelt.

Grid Middleware Anwendung, die einen zentralen Zugriff auf die lose gekoppelten heterogenen Systeme ermöglicht.

Grid Ressource Computer, eine Software, Daten oder weitere Ressourcen, die gemeinsam bei dem Grid Computing genutzt werden.


Hub Komponente der Middleware Infrastructure, die Routes und Middleware Consumer verwaltet sowie Aufgaben einer Service Registry erfüllt.

Hub & Spoke Netzwerkarchitektur, die den Datenaustausch zwischen Service Consumers und Service Providers über einen zentralen Knotenpunkt realisiert.

Hub Client Klasse, die Bestandteil der Middleware APIs ist und einen Zugriff auf die Schnittstellen eines Hubs ermöglicht.

Hub Management Interface Benutzeroberfläche zur Verwaltung eines Hubs.

Hub Storage Klasse, die Bestandteil der Middleware APIs ist und Routes lokal zwischen speichert.

Human Resource Begriff aus der Betriebswirtschaft, der das Personalwesen bezeichnet.


Independently Deployed Applications In Open edX werden Microservices als Independently Deployed Applications bezeichnet.

Indexierung Begriff aus dem Information Retrieval, der die Verschlagwortung und Aufbereitung von Sachverhalten (Daten) bezeichnet.

Internationalized Resource Identifier ein internationaler Standard des Uniform Resource Identifier, der zur Identifikation von Ressourcen dient.

Interoperabilität unterschiedliche Systeme arbeiten möglichst nahtlos zusammen.

Inversion of Control Umsetzungsparadigma, das den Kontrollfluss von Unterprogrammen in einem System an übergeordnete Systemkomponenten abgibt.

Java Naming and Directory Interface Schnittstellen in Java zum Ablegen und Abrufen von Objektreferenzen und Daten anhand eines Namens.

Java-EE Java Softwarearchitektur zur Erstellung mehrschichtiger Unternehmenssoftware, wie beispielsweise im Rahmen dieser Arbeit die Middleware Infrastructure.

JavaScript Object Notation ein mensch- und maschinenlesbares Datenformat, wird meistens zum Datenaustausch zwischen verschiedenen Systemen verwendet.

Joint Degree gemeinsamer Rahmenplan von zwei Universitäten, um zwei akademische Abschlüsse in einem kürzeren Zeitraum zu erlangen.

JS Input wird im Zusammenhang mit der Open edX-Plattform dazu verwendet, Lerninhalte über eigene Javascript-Applikationen einzubinden.

Learning Content Management System Anwendung, die Kurse, Nutzer und Lerninhalte verwaltet.

Learning Locker eine Learning Analytics Plattform und im Rahmen dieser Arbeit eine Drittanbieteranwendung.

Learning Management System Anwendung, die Kurse und Nutzer verwaltet.

Learning Record Store Datenbank zum Speichern von Lernaktivitäten.

**Linked Data** Bereitstellen von strukturellen Daten, die semantisch miteinander verknüpft werden, mit dem Ziel, die Maschinenlesbarkeit zu verbessern.

**Load Balancing** Anwendung, die bei einer hohen Anzahl an Clientanfragen eine Lastenverteilung auf mehrere parallele Systeme gewährleistet.

**Lookup** Im Rahmen dieser Arbeit beschreibt der Begriff lookup den Prozess zum Aufsuchen eines Data Access Object.

**Machine-to-Machine** maschinelle Kommunikation zwischen mindestens zwei Systemen.

**Message Queue** asynchrone Kommunikation, bei der die Nachrichten sequenziell in einer Warteschlange gehalten werden, bis der Empfänger die Nachricht erhält.

**Microservice** feingranulare Dienstleistung, die entkoppelt Teilaufgaben löst (siehe auch **Orchestration**).

**Middleware** Anwendung, die zwischen verschiedenen Systemen vermittelt.

**Middleware API** Komponente der **Middleware Infrastructure**, die Vermittlungsschnittstellen zwischen Clientanwendung und Drittanbieteranwendungen bereitstellt.

**Middleware Consumer** Entwickler von Clientanwendungen, die Zugriff auf mindestens eine Middleware API haben.

**Middleware Infrastructure** gesamte Architektur, bestehend aus Hubs und Middleware APIs.

**Middleware Infrastructure Provider** Dienstleister, der Hubs und Middleware APIs zur Verfügung stellt.

**MongoDB** dokumentenorientierte NoSQL-Datenbank.

**Monolith** selbstverwaltende Anwendung, die intern starke Abhängigkeiten aufweist.

**MOOC** Kurs, bei dem eine hohe Anzahl an Teilnehmern ermöglicht wird.

**Moodle** ein Learning Management System und im Rahmen dieser Arbeit eine Drittanbieteranwendung.

**Multi Hub Management** Verwalten mehrerer Hubs über die Benutzeroberfläche.

**MySQL** Datenbankverwaltungssystem, das eine relationale Datenbank verwendet.

**Open-Source-Software** entsprechend der Lizenzbestimmungen ist der Quellcode der Software frei zugänglich.

**Orchestration** Kombination und Interaktion von feingranularen Dienstleistungen, um komplexere Anwendungsfunktionalitäten zu erfüllen und den modularen Aufbau einer Anwendung fördert.

**Overhead** Daten, die nicht zu den Nutzdaten gehören (Mehraufwand).

**Personal Learning Environment** persönliche Gestaltung der eigenen Lernumgebung. Im Rahmen dieser Arbeit wählt ein Nutzer aus technischer Sicht seine eigenen Webservices.

**Publish and Subscribe** Entwurfsmuster, bei dem Systemkomponenten selektiv (Subscriber) über Nachrichten von anderen Systemkomponenten (Publisher) benachrichtigt werden, mit dem Ziel, eine lose Kopplung und Skalierbarkeit in einem System zu erreichen.

**Ramp-Up** Begriff aus der Wirtschaftswissenschaft und der Wirtschaftsinformatik, der sich auf die Anlaufphase eines Produktes bezieht.

**Read-Only** keine Schreibrechte auf eine Datenhaltungsschicht.

**Recommendation Engine** Komponente, die im Rahmen des SLHw-Projekts Lernempfehlungen für Lernende generiert.


**Reusable Learning Objects** wiederverwendbare Lernobjekte, die mithilfe entsprechender Metadaten vielseitig miteinander kombiniert werden können.

**Rich Client Application** Clientanwendung, die Anwendungsslogik direkt integriert.

**Route** Beschreibung eines Verbindungsweges von einer Clientanwendung bis zur gewünschten Drittanbieteranwendung.

**Run-time** Zeitpunkt, zu dem ein Computerprogramm ausgeführt wird.

**Service** Dienstleistung, die von einer Anwendung erbracht wird.

**Service Broker** Dienstleistungsvermittler, der einen Datenaustausch zwischen verschiedenen Systemen ermöglicht.

**Service Consumer** Konsument von Dienstleistungen und im Rahmen dieser Arbeit Clientanwendungen.

**Service Provider** Anbieter von Dienstleistungen und im Rahmen dieser Arbeit Drittanbieteranwendungen.

**Service Registry** Register beziehungsweise eine Datenablage für Beschreibungen von Dienstleistungen und im Rahmen dieser Arbeit für Routes vorgesehen.

**Service Requester** anfragender Akteur einer Dienstleistung, meist Service Consumer.
Serviceorientierte Architektur: Dienstorientierte Architektur, um eine interoperable Kommunikation in verteilten Systemen zu gewährleisten.


Stacktrace: Ausgabe zur Zurückverfolgung von verschachtelten Funktionsaufrufen, kann zur Fehleranalyse verwendet werden.

Stand-Alone Mode: Betriebsmodus des Applikationsservers Wildfly, der besagt, dass die Konfiguration, die Deployments und Verzeichnisse nur für einen einzigen Server verwendet werden.

Stringify: Serialisierung eines Datenmodells in eine Zeichenkette (String).

Third Party Module: Komponente der Middleware Infrastructure, die einen Datenzugriff und eine einheitliche Datentransformation für eine Drittanbieteranwendung bereitstellt.

Trade-off: Gegenläufige Abhängigkeit, die aus sagt, dass ein Umstand sich verbessert und ein anderer sich verschlechtert.

User Credentials: Zugangsdaten eines Endnutzers von Clientanwendungen.

Virtual File: Datei, die sich im Dateisystem des Applikationsservers Wildfly befindet.

Virtual File System: Dateisystem des Applikationsservers Wildfly.


Web Application Archive: Dateiformat, das die Webanwendung zum Ausliefern beinhaltet.


Wildfly: Applikationsserver, der im Rahmen dieser Arbeit dafür verantwortlich ist, Hubs und Middleware APIs bereitzustellen.

XAMPP: Eine Apache Distribution, die verschiedene Software ohne eine aufwendige Installation bereitstellt. Zu der Software gehört ein Apache Webserver, PHP, MariaDB und Perl.

XBlocks: sind in der Open edX-Plattform Module, aus denen sich ein Kurs zusammensetzt.
1 Einleitung

Diese Arbeit beschäftigt sich mit der interoperablen und serviceorientierten Nutzung von E-Learning-Anwendungen. Der Begriff Interoperabilität bedeutet, dass unterschiedliche Systeme möglichst nahtlos zusammenarbeiten, wodurch die Nutzung von E-Learning-Angeboten vereinfacht wird.¹

1.1 Motivation


Im Rahmen des fMOOC-Projektes nicht vorgesehen, aber bereits angestrebt, war eine Bereitstellung von standardisierten Schnittstellen für Vitaldaten eines Nutzers. Jeder Anbieter eines Fitnesstrackers verfügt jedoch über unterschiedliche Schnittstellen zum Abrufen dieser Daten. Diese Dienstleistung beziehungsweise die Art der Vitaldaten ist bei nahezu allen Anbietern identisch. Es stellt sich grundsätzlich die Frage, warum für jeden Anbieter ein hoher Implementierungsaufwand eingeplant werden sollte, wenn die Dienstleistung und die Daten nur technisch heterogen sind, aber funktional kaum einen Mehrwert beziehungsweise Unterschied aufweisen. Die Integration von möglichst vielen Anbietern ist attraktiv, da durch die Unterstützung verschiedener Fitnesstracker eine hohe Marktabdeckung erreicht werden kann.


¹Duden Stand Februar 2017
1.2 Problemstellung


Werden zusätzliche Anforderungen für ein lebenslanges Lernen miteinbezogen, so steigt die Komplexität erneut. In diesem Fall ist es nötig, zusätzlich mehrere Bildungseinrichtungen zu berücksichtigen. Dies hat zur Folge, dass eine Clientanwendung regelmäßig an neue Anforderungen angepasst werden muss und ein sehr hoher zeitlicher Aufwand bei der Wartbarkeit entsteht.


1.3 Zielsetzung

Das Ziel dieser Arbeit ist die Entwicklung einer Middleware Infrastructure (MI), die für verschiedene Anwendungen zentrale und einheitliche Schnittstellen bereitstellt.


Datenhoheit des Nutzers: Im Unterschied zu serviceorientierten Ansätzen, bei denen eine Kommunikation zwischen Drittanbietern ermöglicht wird, ist in dieser Arbeit ein direkter Datenaustausch zwischen Bildungseinrichtungen zu vermeiden. Die MI vermittelt die Daten aus einer Bildungseinrichtung stets an eine Clientanwendung. Erst die Clientanwendung ermöglicht eine Aggregation der Daten zwischen Bildungseinrichtungen. Auf diese Weise wird die Datenhoheit eines Nutzers begünstigt.


Branchenübergreifend: Langfristig sollen Middleware-Schnittstellen für unterschiedliche Branchen bereitgestellt werden. Zu diesem Zweck wird neben einer serviceorientierten Herangehensweise zugleich die Unabhängigkeit von Prozessen in Form von Microservices vorangetrieben (domänenspezifische API).

1.4 Methodik

In dieser Arbeit wird eine softwareorientierte Methodik verfolgt und dabei zunächst die E-Learning-Branche betrachtet. Hierbei soll ermittelt werden, ob der Bedarf und die Voraussetzungen für eine interoperable Nutzung gegeben sind.

Im weiteren Verlauf werden mehrere Architekturkonzepte der E-Learning-Branche betrachtet. Dieses Vorgehen soll es ermöglichen, die wesentlichen Funktionalitäten der Branche zu ermitteln. Es werden bisher ungelöste Probleme aufgezeigt, die im Rahmen dieser Arbeit mit der geplanten MI gelöst werden können.

Es erfolgt eine Analyse der zu entwickelnden Software. In diesem Zusammenhang wird ein branchenübergreifender Produkteinsatz mit Bezug zu den Recherchen der E-Learning-Branche eruiert.

Da langfristig eine abstrakte MI für mehrere Branchen vorgesehen ist, wird an dieser Stelle frühzeitig das berufliche Umfeld im Sinne der Fortbildung miteinbezogen. Dies bekräftigt außerdem die Relevanz der Datenhoheit eines Nutzers.

Auf Basis des branchenübergreifenden Produkteinsatzes werden in Anlehnung an ein Lastenheft die Anforderungsspezifikationen extrahiert. Diese dienen als Grundlage für den Systementwurf, aus dem ein Prototyp entwickelt wird.
Der Prototyp wird im Rahmen des Projekts *Smart Learning – Medieneinsatz in der handwerklichen Weiterbildung* (SLHw) – in drei realen Anwendungsszenarien erprobt. In der Evaluation der MI werden der fiktive Produkteinsatz, die Erfahrungen während der Entwicklung und die realen Anwendungsszenarien ausgewertet.

Ziel ist es, den Mehrwert des Prototyps zu ermitteln und eine alternative Denkweise gegenüber *Monolithen* und *Insellösungen* anzuregen.

### 1.5 Aufbau der Arbeit

**Related Work:** Dieses Kapitel verdeutlicht den Wandel der E-Learning-Branche sowie mit dieser Arbeit verwandte Architekturen und das Potenzial von serviceorientierten Ansätzen.

**Systemanalyse:** Dieses Kapitel beschreibt einen umfangreichen Produkteinsatz und die daraus resultierenden Anforderungsspezifikationen. Zudem werden grundlegende Fachbegriffe der zu konzipierenden Architektur angedeutet.

**Systementwurf:** Dieses Kapitel zeigt den Aufbau der MI. Für jede Komponente werden die zu lösenden Teilprobleme identifiziert und anschließend in einem finalen Systementwurf präsentiert.

**Implementierung:** Dieses Kapitel basiert auf dem Systementwurf und präsentiert eine logische Codestruktur, bestehend aus funktionalen Modulen. Auf diese Weise wird eine lose Kopplung von Funktionalitäten realisiert. Relevante Aufgaben eines funktionalen Moduls werden mit Codeausschnitten ergänzt.

**Evaluation:** Dieses Kapitel stellt reale Anwendungsszenarien vor. Zudem erfolgt eine kritische Betrachtung des Prototyps. Es werden sowohl die qualitativen (Kernziele) als auch die quantitativen (Performance) Softwarekriterien ausgewertet.

**Zusammenfassung:** In diesem Kapitel werden die wichtigsten Ziele in einem Fazit zusammengefasst. Ein Ausblick beschreibt bisher ungelöste Funktionalitäten und weitere Potenziale der MI.
2 Related Work


2.1 Die Bedeutung von Learning-Management-Systemen (LMS)


Coates et al. (2005) beschreibt ein LMS als unternehmensweites und internetbasiertes System, wie zum Beispiel WebCT und Blackboard, die eine Vielfalt an pädagogischen Elementen sowie Werkzeuge zur Kursverwaltung beinhalten. Diese Systeme erlauben virtuelle Lernumgebungen für Präsenzstudenten (engl. campus-based students), aber auch für die Entwicklung von vollständigen Online-Kursen.


2.2 Die heterogene Systemlandschaft von LMS

LMS wurden durch die multimedialen Möglichkeiten des Internets seit den neunziger Jahren vorangetrieben (Coates et al., 2005). Es haben sich auf dem Markt viele LMS-Anbieter etabliert, wie die Abbildung 1 [S.18] verdeutlicht.
Abbildung 1: Share of LMS market worldwide in 2013, by vendor (Statista, 2013)

Im Jahr 2005 gab es mehr als 250 Anbieter für kommerzielles E-Learning und über 40 Open-Source-Software-Angebote (OSS). Die bekanntesten OSS sind Modular Object-Oriented Dynamic Learning Environment (Moodle), Ilıas, Eduplone, SAKAI und WebCT (Zedan and Al-Ajlan, 2005).


Tabelle 1: E-Learning Standards (Eigene Tabelle)

<table>
<thead>
<tr>
<th>IMS Global Learning Consortium (IMS)</th>
<th>IMS Metadata, IMS Content Packaging, IMS Question &amp; Test Interoperability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced Distributed Learning Network (ADLNet)</td>
<td>Sharable Content Object Reference Model (SCORM)</td>
</tr>
</tbody>
</table>


### 2.3 Eine historische Betrachtung von LMS


Zudem wurden durch die Modularisierung Webservices ermöglicht, die LMS- und LCMS-Funktionalitäten extern zur Verfügung stellten.

The LMS vendor of old will no longer sell monolithic, one-size-fits-all solutions, but rather interoperable platforms and a range of e-learning services, letting consumers choose the right combination of services for their requirements. (Dagger et al., 2007)


Lernen ermöglicht wird. Auch die Offline-Verfügbarkeit von digitalen Lerninhalten und das mobile Lernen sind wichtige Faktoren.

Sowohl pädagogische als auch technologische Anforderungen sind bei der Konzeption eines PLE von Bedeutung. In dieser Arbeit richtet sich der Fokus auf die technologischen Anforderungen.

Neben der Vernetzung von Institutionen, wie zum Beispiel Google oder Facebook, könnten PLEs zusätzlich in der Lage sein, passende Lerninhalte für den Nutzer zu ermitteln. Mit einer personalisierten SOA, die verschiedene Webservices zentral für den Nutzer bereitstellt, wäre dies möglich. Die zu konzipierende MI verfolgt das Konzept, wie bei dem PLE, Daten eines Nutzers durch Einwilligung an eine weitere Institution zu transferieren.

Der zweite Trend, der gegensätzlich erscheinen mag, beschäftigt sich mit der Bereitstellung von Online-Kursen, die sich durch eine sehr hohe Erreichbarkeit und eine massive Anzahl an Nutzern auszeichnen. In diesem Zusammenhang wird von massive open online courses (MOOC) gesprochen. Die MOOC unterteilen sich in cMOOC und xMOOC.

Das c in cMOOC steht für connectivism, zu Deutsch Konnektivismus. Hierbei steht als Lerntheorie das soziale Gefüge und der Austausch von Informationen im Vordergrund.


Connectivism presents a model of learning that acknowledges the tectonic shifts in society where learning is no longer an internal, individualistic activity. (Siemens, 2005)

Das x in xMOOC steht hingegen für eine Erweiterung (engl. extensible) des MOOC-Konzepts. Anders als bei dem Konnektivismus werden die traditionellen Ideen des Lehrens und Lernens um moderne Informationstechnologien erweitert (Peters et al., 2015).

### 2.4 Das Potenzial von serviceorientierten Ansätzen in der E-Learning-Branche

In diesem Kapitel werden einige grundlegende Potenziale von serviceorientierten Ansätzen im Bereich der E-Learning-Branche aufgezeigt.


Technologisch sind die bisherigen monolithischen Systeme nicht in der Lage, ohne hohen Aufwand neue Funktionalitäten bereitzustellen, was die Individualisierung von E-Learning-Angeboten einschränkt. (vgl. Frankfurth and Schellhase, 2006)

Organisatorisch führt die mangelnde Interoperabilität bei interuniversitären Lehrkooperationen zu einer unnötigen Mehrarbeit, da die einzelnen Systeme nicht miteinander kommunizieren können. (vgl. Frankfurth and Schellhase, 2006)


Im Zusammenhang mit SOAs wird in der E-Business-Branche vermehrt von *Microservices* gesprochen, ein Konzept, das auch in der E-Learning-Branche Anwendung

<table>
<thead>
<tr>
<th>Platform</th>
<th>Based in</th>
<th>Estd.</th>
<th>Business Model</th>
<th>Courses</th>
<th>Partners</th>
<th>Courses / Partners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coursera</td>
<td>US</td>
<td>2012</td>
<td>for-profit</td>
<td>571</td>
<td>100</td>
<td>5.7</td>
</tr>
<tr>
<td>open edX</td>
<td>US</td>
<td>2012</td>
<td>non-profit</td>
<td>200</td>
<td>53</td>
<td>3.8</td>
</tr>
<tr>
<td>FutureLearn</td>
<td>UK</td>
<td>2012</td>
<td>for-profit</td>
<td>53</td>
<td>40</td>
<td>1.3</td>
</tr>
<tr>
<td>Udacity</td>
<td>US</td>
<td>2011</td>
<td>for-profit</td>
<td>39</td>
<td>11</td>
<td>3.5</td>
</tr>
</tbody>
</table>


In der Regel benötigen Webservices in monolithischen LMS noch immer systeminterne Daten, um funktionieren zu können. Eine Änderung an dem System kann aufgrund interner Abhängigkeiten unter Umständen dazu führen, dass mehrere Webservices betroffen sind.

Nach Fowler and Lewis (2015) wird ein Microservice soweit losgelöst, dass die Funktionalitäten bis zur Datenerhaltungsschicht getrennt werden. Demnach ist ein Microservice auf einem so hohen Grad unabhängig, weshalb ein Deployment auf verschiedenen Plattformen und Prozessen stattfinden kann.

Ein Nachteil, der in diesem Zusammenhang genannt wurde, ist, dass die prozessinterne Kommunikation in einem Monolithen eine bessere Performance aufweist als die Kommunikation in einer Microservice-Architektur.
Lösen lässt sich dieses Problem durch grobkörnige remote APIs, was möglicherweise die Bedienung und Wartbarkeit negativ beeinflusst. Ein entscheidender Vorteil von einer Microservice-Architektur ist, dass ein Hinzufügen oder Entfernen von neuen Services kein erneutes Deployment des vollständigen Systems erfordert.


### 2.5 Architekturkonzepte von serviceorientierten Ansätzen in der E-Learning-Branche

SOAs verfolgen das find-bind-execute-Paradigma für die infrastrukturelle Kommunikation. Die Abbildung 3 [S.23] verdeutlicht die Grundidee von SOAs.

- **Service Consumer** stellt eine Anfrage an den **Service Broker**, um eine geeignete Servicebeschreibung eines **Service Providers** zu erhalten (find). Jeder **Service Provider** wird über die **Service Registry** registriert und ist für **Service Consumer** verfügbar. Sobald ein **Service Consumer** eine passende Servicebeschreibung von dem **Service Broker** erhalten hat, beginnt der Verbindungsaufbau. Die Verbindungsdaten des **Service Providers** werden gebunden (bind). Zuletzt erfolgt die Ausführung, bei dem der **Service Consumer** den **Service Provider** aufruft (execute).

Bei der Konzeption von SOAs sind standardisierte Schnittstellen von hoher Bedeutung, um eine Kommunikation über Webservices zu ermöglichen.
Abbildung 4: Generations of learning management systems (Dagger et al., 2007)


Das *Application Layer* besteht aus Tools und Systemen, die dem Nutzer verschiedene *Application Services* aus dem *Application Services Layer* präsentiert.


Das *Common Services Layer* stellt Services bereit, die von den *Application Services* verwendet werden. Ein *Common Service* kann einen anderen *Common Service* nutzen. Bei diesem könnte es sich zum Beispiel um einen Authentifizierungsservice handeln.


<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassAdministration</td>
<td>the management of classes</td>
</tr>
<tr>
<td>ContentManagement</td>
<td>management of learning content</td>
</tr>
<tr>
<td>LearnerProgressionManagement</td>
<td>management of learning experiences</td>
</tr>
<tr>
<td>MetaDataManagement</td>
<td>management of meta-data resources</td>
</tr>
<tr>
<td>DigitalRepositoryManagement</td>
<td>management of digital repositories</td>
</tr>
<tr>
<td>ProfileManagement</td>
<td>management of a learner’s profile, lifelong learning log, etc.</td>
</tr>
<tr>
<td>EnterpriseServices</td>
<td>management of course enrolments and learning activities</td>
</tr>
</tbody>
</table>

Es gibt viele Möglichkeiten, eine SOA für die E-Learning-Branche zu entwickeln. Das IAF beschreibt, wie ein LCMS in Services aufgeteilt werden kann.

Das ELF war bis etwa 2007 eine internationale Bemühung, einen serviceorientierten Ansatz für die Integration von Systemen im Bereich des Lernens, der Forschung und der Bildungsadministration zu entwickeln. (ELF, n.d.)

Für die Konzeption der MI werden die Services der MWA an das IAF beziehungsweise die ELF Application Services angelehnt.

von *Grid-Computing* lässt sich nach einer neuen Definition von Foster and Kesselman (2003) in die drei folgenden Bestandteile zerlegen:

Es wird eine gemeinsame Nutzung von Ressourcen angestrebt, indem der Zugriff auf Computer, Software und Daten erfolgt.

   The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. (Foster and Kesselman, 2003)

Der Zugriff beziehungsweise das Teilen von Ressourcen wird von den Providern (Anbieter) und Consumern (Konsumenten) kontrolliert.

   This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. (Foster and Kesselman, 2003)

Eine Menge von Individuen oder Institutionen bilden nach den Richtlinien der gemeinsamen Nutzung von Ressourcen eine *Virtuelle Organisationen* (VO).

   A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization. (Foster and Kesselman, 2003)


In einem Grid wird zudem eine *Grid Middleware* eingesetzt, um Nutzern und Applikationen einen Zugriff auf die *Grid Ressourcen* zu erleichtern. Aus Sicht des Nutzers wird die Komplexität bezüglich des Managements, der Richtlinien und der Zugriffsart verborgen. (vgl. Ivanović and Jain, 2014)

Das *Globus Toolkit*, entwickelt von der *Globus Alliance*, ist eine *Grid Middleware*, die mithilfe verschiedener *Toolkits* und *Core Services* die Entwicklung einer *Grid Application* ermöglicht.
In der Abbildung 6 [S.27] wird die grid-basierte Architektur *E-Learning Grid* dargestellt.

Abbildung 6: Layered architecture of e-Learning Grid (Ivanović and Jain, 2014)

Das *Application Layer* beinhaltet die Benutzeroberflächen für die Lernenden.

Das *Learning Layer* besteht aus LMS Services und kommuniziert direkt mit der *Grid Middleware*.


Die zu konzipierende MI orientiert sich von der Grundidee an dem Paradigma des *Grid Computing* und der Verwendung einer *Grid Middleware*.


In diesem Szenario sind nur die Systeme LCMS und LMS in der Lage, eine Dienstleistung zu beziehen und anzubieten. Das Autorensystem (Authoring Tools) interagiert mit Lerninhalten aus einem LCMS über einen *Content Service Requester*.

Es gibt zwei *Service Broker* Komponenten, die in der Lage sind, LCMS und LMS als Servicebeschreibungen anzubieten. Die *Content Service Discovery Agencies* vermit-
n Servicebeschreibungen zu LCMS-Service Providern. Die Learner Service Discovery Agencies liefern hingegen Beschreibungen zu LMS-Service Providern.

Abbildung 7: Service architecture of e-learning system (Liu et al., 2003)


Abbildung 8: Joint Degrees in E-Learning Systems (Aguirre et al., 2008)
Es handelt sich hierbei um ein Konzept, bei dem zwei Universitäten vertraglich einen gemeinsamen Lehrplan ermöglichen. Die Kursinhalte aus den E-Learning-Systemen der Universitäten werden hierbei verknüpft.

Die Joint Degree Services verwalten die benötigten Rahmenbedingungen, wie zum Beispiel die Dauer und die Bildungsvoraussetzungen.

Neben dem Service Provider und Service Broker wird noch ein Identity Provider vorgesehen, der die Validierung und die Identität der Nutzer überprüft.


Die personalisierte SOA von Dagger et al. (2007) beschreibt exemplarisch, wie ein Mitarbeiter einer Firma durch Integration firmenspezifischer Datenbanken im Bereich Human Resource (HR) und Customer-Relationship-Management (CRM) eine gefilterte Menge an Lerninhalten erhält, die für die Bearbeitung eines Auftrags benötigt werden.

In der Abbildung 9 [S.29] sind die für diesen Prozess benötigten Schnittstellen abgebildet.

![Abbildung 9: Dynamic personalized E-Learning scenario (Dagger et al., 2007)](image)

Der Mitarbeiter verbindet sich über das Lernportal mit der Technical Knowledge Database (TKB), um an die für den Auftrag benötigten Lerninhalte zu gelangen [1].

Über den Federated User Modeling Exchange Service (FUMES) wird das aktuelle Lernprofil aus den von dem Mitarbeiter genutzten LMS abgefragt [2].

Das Adaptive Contextual Portal (ACP) kann das Lernprofil auf Basis von firmeninternen Datenbanken anpassen [3]. Durch HR wird der Beruf des Mitarbeiters ermittelt.

Sofern die TKB im Learning Repository nicht die nötigen Inhalte zur Verfügung stellt, wird der *Open Corpus Content Service* (OCCS) abgefragt [4]. Bei dem OCCS handelt es sich um einen Web-Crawler, der das Internet sowie digitale Repositories durchsucht und die gewünschten Lerninhalte für das TKB indexiert.


Abbildung 10: Open edX MOOC Architecture (EdX, 2016)

Die *EdX Platform* beinhaltet sowohl das LMS als auch die Autorenwerkzeuge (Studio) zur Erstellung von Lerninhalten.


IDAs bieten die Möglichkeit, eigenständige Services in die Architektur zu integrieren. Hierbei könnte es sich zum Beispiel um ein Forum handeln. Über eine API wird durch das LMS der *comments Service* (Forum Service) abgerufen. Aufgrund der potenziell hohen Anzahl an Kursteilnehmern und den damit verbundenen aufwendigen Berechnungen sind einige Aufgaben (engl. tasks) über *Message Queues* zu verarbeiten.
Die Events für die Learning Analytics (LA) sind in einem JSON-Format auf Amazon S3 abgelegt. Hadoop Jobs (MapReduce) aggregieren die Events und legen die Ergebnisse in einer MySQL-Datenbank ab. Der externe Zugriff erfolgt über eine REST API. Es ist anzumerken, dass SOAs den Ansatz verfolgen, Services weitestgehend zu kapseln. Die MOOC fokussieren sich aufgrund der hohen Nutzerzahlen zusätzlich bei der Datenerhaltung, dem Eventhandling und dem Tracking auf skalierbare Web-Technologien.

2.6 Zusammenfassung


Es werden zwei Arten von potenziellen Drittanbieteranwendungen behandelt. Zum einen wird das lebenslange Lernen durch eine Personal Learning Environment (PLE) beschrieben, zum anderen steigt auch der Bedarf, eine hohe Erreichbarkeit von Kursen zu ermöglichen, wie bei MOOCs zu beobachten ist.

Auf Basis der vorgestellten Drittanbieteranwendungen und SOAs können in dem nächsten Kapitel Anforderungen an die zu konzipierende Middleware Infrastructure gestellt werden.

Bei den vorgestellten Architekturkonzepten ist das Paradigma des Grid Computings und der Einsatz der Grid-Middleware vergleichbar mit der zu konzipierenden MI.
3 Systemanalyse

In diesem Kapitel erfolgt eine Systemanalyse des zu konzipierenden Systems. Es steht die Frage im Vordergrund, wie sich das System im Vergleich zu den Konzepten aus dem Kapitel Related Work unterscheidet und welches Problem beabsichtigt wird zu lösen.

Es folgt ein fiktives Szenario für einen branchenübergreifenden Produkteinsatz. Hierbei sollen die Potenziale der MI verdeutlicht werden.

Abschließend wird eine Anforderungsspezifikation ausgearbeitet, um die konkreten Anforderungen in einem Systementwurf umsetzen zu können.

3.1 Das zu lösende Problem und der Mehrwert des Systems


Unter der Berücksichtigung von unterschiedlichen Anwendungsszenarien haben alle Architekturen Vor- und Nachteile, unabhängig von der Beobachtung, dass SOAs aktuell vermehrt thematisiert werden.


### 3.2 Ein branchenübergreifender Produkteinsatz

Ein branchenübergreifender Produkteinsatz eignet sich, um die Grundidee und Flexibilität der MI zu verdeutlichen. Zudem können feingranular Anforderungsspezifikationen ermittelt werden.


Abbildung 11: Abstract Middleware Infrastructure – MI (Eigene Darstellung)
Die MI soll ein E-Learning-Angebot ermöglichen, das neben den Bildungseinrichtungen (Universitäten) zusätzlich Daten aus dem beruflichen Umfeld und einer Bibliothek bereitstellt.

In dem Kapitel *Related Work* wurden einige Konzepte für die E-Learning-Branche vorgestellt, die auch in den fiktiven Produkteinsatz integriert werden.


Eine elementare Voraussetzung bei diesem Szenario ist, dass der Middleware Infrastructure Provider (MIP) Zugriffsdaten zu allen *Drittanbieteranwendungen* erhält. Das bedeutet, alle Institutionen müssen dem MIP vertrauen. Eine Möglichkeit könnte eine Verstaatlichung des MIP sein, was zu einem Bundesnetzwerk (*federal network*) führt. Der MIP entscheidet, welche *Drittanbieteranwendungen* von einer *Clientanwendung* abgerufen werden können. Innerhalb der MI liegen die Verbindungsinformationen für *Drittanbieteranwendungen* in der *Service Registry*.


Von allen Institutionen kann das Nutzungsverhalten ermittelt werden. Für diesen Zweck wird die LRS API bereitgestellt. In dem aktuellen Szenario verwenden die Universitäten die Anwendungen *Piwik* und *Learning Locker*.


Ein Kriterium bei der Konzeption ist, dass eine Clientanwendung die Services der MWAs verknüpft. Es soll verhindert werden, dass personenbezogene Daten direkt zwischen Drittanbieteranwendungen der Institutionen ausgetauscht werden. Aus diesem Grund bezieht eine Clientanwendung alle nötigen Daten.


### Tabelle 4: Mapping between Third Party and Middleware API (Eigene Tabelle)

<table>
<thead>
<tr>
<th>Third Party</th>
<th>LRS</th>
<th>USR</th>
<th>COM</th>
<th>LCMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moodle</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Blackboard</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Moodle Forums</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Blackboard Forums</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Human Resources (HR)</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Repository</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Learning Locker</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Piwik</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
3.3 Die Anforderungsspezifikationen als Grundlage für den Systementwurf

In diesem Abschnitt erfolgt eine Unterteilung der Anforderungsspezifikationen nach Muss-Kriterien, Soll-Kriterien, Kann-Kriterien und Abgrenzungskriterien.

3.3.1 Muss-Kriterien


A_M3: Beliebige Drittanbieteranwendungen müssen an das System angebunden werden. Um die Anforderungen der Services erfüllen zu können, muss das System bei der Anbindung von Drittanbieteranwendungen in der Lage sein, unterschiedliche Schnittstellen zu vereinen.


3.3.2 Soll-Kriterien

_A_S1:_ Eine Institution soll für Kommunikationswege eingeschrieben werden. Daraus folgt, dass mehrere Institutionen bei Bedarf auf dieselben Kommunikationswege zugreifen.

_A_S2:_ Die Anbindung von *Drittanbieteranwendungen* soll in der Nachnutzung auch in Form von Plugins realisiert werden können. Die Plugins werden als *Third Party Modules* (TPM) bezeichnet.

_A_S3:_ Das System besitzt eine zentrale Anlaufstelle für potenziell mehrere Institutionen. Aus diesem Grund soll eine Institution eindeutig identifiziert werden können. Die Institutionen werden als *Middleware Consumer* bezeichnet.

_A_S4:_ Der Zugriff auf die Kommunikationswege soll durch die Authentifizierung von Institutionen abgesichert werden.

_A_S5:_ Bezugnehmend auf Datenschutzrichtlinien soll eine Nutzer-Authentifizierung mit *Drittanbieteranwendungen* ermöglicht werden, ohne Zugangsdaten an das System zu übermitteln. In diesem Fall ist eine direkte Kommunikation mit der *Drittanbieteranwendung* möglich.

3.3.3 Kann-Kriterien

_A_K1:_ Die Verbindungsart zu *Drittanbieteranwendungen* kann ausgetauscht werden. Die Komponente, die für die Verbindungslogik verantwortlich ist, wird als *Connector* bezeichnet.

_A_K2:_ Die Konfiguration von Institutionen und Kommunikationswegen kann über ein *Management Interface* ohne großen Aufwand vorgenommen werden.

_A_K3:_ Eine Skalierung des Systems kann durch ein verteiltes und unabhängiges Deployment der Systemkomponenten gewährleistet werden.

3.3.4 Abgrenzungskriterien

_A_AB1:_ Das System wird nicht für die Bereitstellung eigener Dienstleistungen konzipiert. Es wird nur eine Transformation der eingehenden Daten vorgenommen, um einheitliche Standards aus Sicht einer *Clientanwendung* zu gewährleisten.

_A_AB2:_ Die standardisierten Schnittstellen haben nicht das Ziel, alle ursprünglichen Funktionalitäten in vollem Umfang zu unterstützen. Grundsätzlich müssen die elemen-
taren Funktionen einer Dienstleistung sichergestellt werden. Es entsteht ein trade-off, bei dem eine interoperable Nutzung der Verwendung von spezifischen Funktionalitäten einer Drittanbieteranwendung vorgezogen wird.

3.4 Zusammenfassung

Zusammenfassend ist festzuhalten, dass Anbieter in der E-Learning-Branche gängige Standards nicht zuverlässig unterstützen, was bei der Interoperabilität zu Problemen führt.

Aufgrund verschiedener Anforderungen hinsichtlich der Integration einer E-Learning-Anwendung ist eine Softwarelösung erforderlich, die für das Adaptieren von Drittanbieteranwendungen verantwortlich ist.

Es wurde ein fiktives Szenario vorgestellt, bei dem dieses Problem berücksichtigt wird. Zudem wurde ein branchenübergreifender Einsatz für das E-Learning angedeutet, um das Potenzial der vorgesehenen MI zu verdeutlichen.

Neben den Anforderungen, die eine Nutzung von PLE und Joint Degrees miteinbeziehen, wurde eine zentrale Kontrollinstanz eingeführt (Middleware Infrastructure Provider). Auf diese Weise könnte eine staatliche Instanz (federal network) Datenschutzrichtlinien und Kontrollflüsse autoritär vertreten.


Basierend auf dem im Kapitel Related Work angeführten SOA-Paradigma, wurden für den Systementwurf relevante fachliche Begriffe und Anforderungen definiert und konkretisiert.
4 Systementwurf


Im weiteren Verlauf wird jede Komponente im Detail betrachtet, indem die zu lösenden Teilprobleme identifiziert und anschließend in Form eines finalen Systementwurfs präsentiert werden.

Abschließend erfolgt eine Überführung der MI in eine Darstellung speziell für die E-Learning-Branche.

4.1 Die Middleware Infrastructure (MI)


Die Aufgabe der MI ist es, eine Clientanfrage über eine MWA anzunehmen, um diese an einen geeigneten *Endpunkt* (*Drittanbieteranwendung*) zu delegieren.

Im Vordergrund steht hierbei die Interoperabilität, die durch den Einsatz von TPM gewährleistet werden soll. Die Geschäftslogik zum Aufruf des passenden TPMs ist Bestandteil der *DAO Manager*-Implementierung. Der Aufruf eines TPMs wird durch die gestrichelte Linie gekennzeichnet.

Es werden exemplarisch drei verschiedene Domänen (D1, D2, D3) dargestellt. Diese stehen beispielsweise für unabhängige Dienstleistungen oder Branchen.
Eine domänenspezifische MWA verwendet die jeweiligen Datenmodelle (Model) der definierten Domäne. Jedes TPM, das Bestandteil der Domäne ist, implementiert die Adapter auf Basis der vorgegebenen Datenmodelle.

Der Hub als übergeordnete Komponente liefert Informationen, die für den Aufbau einer Verbindung mit einem Endpunkt benötigt werden. Die Informationen sind als sogenannte Route an die entsprechende MWA zu übermitteln. Der Zugriff auf eine Route wird initial durch den Hub geregelt.


### 4.2 Der Hub


Die Grundidee, die auf dieser Einschränkung basiert, ist die Verwendung von festgelegten Verbindungswegen (Route). Diese werden dynamisch über den Hub konfiguriert und durch das anfragende System (Client) ausgewählt.

Im Vergleich zu den SOAs Hub & Spoke (Abbildung 13 [S.41]) und Enterprise Service Bus (kurz. ESB, Abbildung 14 [S.41]) bestimmt die Clientanwendung als Akteur den Dienst eines Drittanbieters. Eine Clientanwendung ist ausschließlich als Service Consumer zu betrachten und kann nicht als Service Provider verwendet werden. Im Detail wird erst durch die Eigenschaften eines Hubs die Identifikation einer SOA ermöglicht. Dies ist auch bei der ESB- und Hub & Spoke-Architektur der Fall.

The ESB is sometimes described as a distributed infrastructure and is contrasted with solutions (such as broker technologies) that are commonly described as hub-and-spoke. In contrast, hub-and-spoke integration solutions seek to centralize control of configuration: routing information, service naming, and so forth. In the Patterns for e-business Process Integration patterns an ESB is classified as a type of bus, which in turn is classified as a type of hub... (Keen et al., 2004, pp. 77)
Die Voraussetzungen und Einsatzmöglichkeiten von SOAs sind sehr verschieden. Bei der in dieser Arbeit vorgestellten MI wird erst durch die Restriktion, in der Service Consumer und Service Provider strikt getrennt werden, eine SOA ermöglicht, in der domänenspezifische MWAs angeboten werden können. In diesem Fall agiert diese aus Sicht einer Clientanwendung wie ein gängiges Backend.

4.2.1 Die Kommunikation zwischen Hub und Middleware API


Abbildung 15: Hub Communication (Eigene Darstellung)


### 4.2.2 Der Aufbau und die Eigenschaften einer Route

Die Route setzt sich aus den fünf Klassen General, Security, Middleware, Module und Endpoint zusammen. Jede Klasse hat in der Gesamtarchitektur eine konsistente Rolle.

In Abbildung 16 [S.43] wird das Klassendiagramm einer Route aufgeführt.
Die Klasse **General** beinhaltet die allgemeinen Parameter `routeID` und `displayName`. Die `routeID` dient der eindeutigen Identifikation einer `Route`. Bei dem `displayName` handelt es sich um eine lesbare Form der `Route`. In erster Linie soll dieses Attribut dazu dienen, in einer *Clientanwendung* den Nutzern die verschiedenen `Routes` präsentieren zu können.

Die Klasse **Security** besitzt alle nötigen Attribute für einen sicheren Verbindungsaufbau mit einem *Endpunkt*. Innerhalb dieser Klasse erfolgt eine weitere Unterteilung in die Klassen **ExternalAuthentication**, **ConsumerCredentials** und **UserCredentials**.

Die Klasse **ExternalAuthentication** ermöglicht eine Authentifizierung direkt mit dem *Endpunkt*. Dieser Use-Case umgeht bei der Authentifizierung die MWA.

Die Klasse **ConsumerCredential** beinhaltet alle Verbindungsdaten, sodass eine MWA (zum Beispiel Administrator oder Manager) gegenüber einem *Endpunkt* authentifiziert werden kann.


Die Klasse **Middleware** liefert der *Clientanwendung* mögliche MWA URLs. In der Regel handelt es sich um eine MWA, die bei dem angefragten *Hub* registriert ist. Technisch kann jede `Route` auch in einer nicht übermittelten MWA genutzt werden. Voraussetzung ist die Registrierung der MWA an den *Hub*. Zudem muss das von der `Route` verwendete TPM vorhanden sein.

Die Klasse **Module** legt fest, welches TPM genutzt werden soll.

Die Klasse **Endpoint** beschreibt über die Attribute `host` und `baseURL`, unter welcher Adresse der Endpunkt abzurufen ist.

### 4.2.3 Das Hub-Management-Interface

Das *Hub Management Interface* ist eine Webapplikation, die eine direkte Kommunikation zwischen Systemadministrator und *Hub* ermöglicht. Über die Benutzeroberfläche können *Middleware Consumer*, `Routes` und `Route Enrolments` verwaltet werden.

Das Management Interface ist durch die Hub-Authentifizierung für ein Multi Hub Management optimiert. Daraus folgt, dass durch die Benutzeroberfläche beliebig viele Hubs verwaltet werden können.

Neben der Hub-Authentifizierung wird ein externer Authentifizierungsserver eingebunden, der für die Authentifizierung von Systemadministratoren verantwortlich ist.

Das responsive Design der Benutzeroberfläche (Abbildung 17 [S.45]) ermöglicht eine Nutzung auf dem Desktop, Tablet und Smartphone.

### Abbildung 17: Showcase Hub Management (Eigene Darstellung)

#### 4.3 Die Middleware API (MWA)


Ein Abgrenzungskriterium ist, nicht alle Funktionen einer Drittanbieteranwendung zu adaptieren. Vielmehr sollten die wichtigsten Funktionen extrahiert werden.

#### 4.3.1 Dynamisierung von Verbindungswegen (Routing)

Ein wichtiges Kriterium bei der Entwicklung der MI war es, für die Nachnutzung verschiedene Domänen unterstützen zu können. Eine Domäne ist mit anderen Domänen nicht verwandt. Jede MWA ist als eigenständige Komponente zu betrachten, wie es auch bei Microservices der Fall ist.

Die Abbildung 18 [S.46] verdeutlicht die Relation zwischen TPMs und den domänen spezifischen MWAs sowie die Möglichkeiten des Routings.

Abbildung 18: Middleware Routing (Eigene Darstellung)

Bei Betrachtung der Abbildung 18 [S.46] wird ersichtlich, dass eine Route für unterschiedliche Anwendungsszenarien optimiert werden kann.

Die Route A verwendet die Services der Domäne D1. Die benötigte Geschäftslogik wird aus dem TPM1 abgerufen. TPM1 ist ebenfalls Bestandteil von D1 und enthält die spezifische Logik für mindestens einen der Services aus der MWA. TPM1 verbindet sich entsprechend der Konfiguration aus Route A mit dem gewünschten Endpunkt E1.

Endpunkte sind bewusst keiner Domäne zugeordnet. Begründen lässt sich dies mit der Gegebenheit, dass ein Endpunkt mehrere Dienstleistungen (Drittanbieteranwendungen) bereitstellen kann, die wiederum einer anderen Domäne angehören können.

Bei Betrachtung der Routes C und F wird dieses Szenario exemplarisch dargestellt. Beide Routes gehören einer anderen Domäne an, verwenden unterschiedliche TPMs und nutzen gemeinsam E3 als Endpunkt.

Eine weitere Randbedingung ist das Teilen eines TPMs und die Verwendung eines weiteren Endpunkts. Die Routes A und B teilen sich das TPM1 und verwenden als
Endpunkte E1 und E2. Es handelt sich in diesem Szenario um eine Dienstleistung, die auf zwei verschiedenen Endpunkten verfügbar ist.

Bei den Endpunkten E1 und E2 könnte es sich um zwei physikalische oder virtuelle Instanzen (Server) handeln.

4.3.2 Die Drittanbieter-Authentifizierung von User und Consumer

Jede Route, die über die MWA verwendet wird, hat die Aufgabe, eine Verbindung zu einer Drittanbieteranwendung aufzubauen. Diese besitzt in der Regel ein Authentifizierungsverfahren. Zudem existieren potenziell Berechtigungshierarchien in Form von Rollen. Um zu gewährleisten, dass eine Route alle benötigten Zugriffsrechte erhält, besteht die Möglichkeit einer clientseitigen (User-) und serverseitigen (Consumer-) Authentifizierung.


Eine Clientanwendung kann bei Anfrage an eine MWA Zugangsdaten eines Nutzers übermitteln. Die übertragenen Zugangsdaten, auch User Credentials (UC) genannt, werden zum Zeitpunkt der Anfrage serverseitig in dem Objekt UserCredential initialisiert und sind somit über die Route innerhalb eines TPM abrufbar. Die Logik des aufgerufenen TPM bestimmt, wie die Daten für die Authentifizierung gegenüber der Drittanbieteranwendung eingesetzt werden können.

Die Consumer Credentials (CC) werden direkt in der Route persistiert und sind Bestandteil der R-priv-Daten. Eine MWA erhält die Routes inklusive der R-priv-Daten. Eine Route wird durch die MWA an ein TPM übergeben und enthält somit auch das Objekt ConsumerCredential.
Sowohl UC als auch CC unterstützen die Authentifizierungstypen **BASIC** (IETF, 2015) und **BEARER** (IETF, 2012). Der aktuelle Entwicklungsstand sieht für UC hinsichtlich der MWA-Schnittstellen nur **BASIC** und **BEARER** vor. Bei CC sind diese frei wählbar, da jedes Modul die Logik für die Authentifizierung selbst bestimmt.

### 4.3.3 Das Individualisieren von Routen


![Abbildung 20: Individualized Routes (Eigene Darstellung)](image)

Die *Route* E1_S1 identifiziert durch die CC eindeutig die gesicherte Sektion S1 des *Endpunkts* E1. In diesem Fall wird nur der *Consumer* authentifiziert. Jeder *Middleware Consumer*, der Zugriff auf die individualisierte *Route* erhält, hat über die *Clientanwendung* vollen Zugriff auf die jeweilige Sektion von E1.

Neben der individualisierten *Route* sind auch Hybrid-Lösungen realisierbar. Daraus folgt, dass die *Route* für eine Sektion individualisiert wurde und zusätzlich ein Nutzer aus der Sektion zu authentifizieren ist. Der Zugriff wird erst nach clientseitiger User-Authentifizierung (UC) vollständig möglich sein.

Die Relevanz und Gewichtung von UC und CC ist abhängig von der *Drittanbieteranwendung* und der Modulimplementierung.

### 4.3.4 Die Bypass-Authentifizierung (User)

Eine weitere Funktionalität der User-Authentifizierung ist der Bypass. In diesem Use-Case soll verhindert werden, dass die *Clientanwendung* direkt den Namen und das
Passwort eines Users an die MWA überträgt. Dies ist jedoch nur möglich, wenn die auf einem Endpunkt angefragte Dienstleistung eine Token-Authentifizierung verwendet. Um bei der Authentifizierung die MWA zu umgehen (Bypass), sendet die Clientanwendung eine Anfrage direkt an die Anwendung des Endpunkts.

Basierend auf der Tatsache, dass jede Dienstleistung eine andere Authentifizierungslogik verwendet, wird ein Lösungsansatz erforderlich, der die Heterogenität aus Sicht der Clientanwendungen verhindert. Aus diesem Grund besitzt jede Route das Attribut `scriptURL` (siehe Abbildung 16), das es ermöglicht, ein Script anzufordern. Dieses Script besitzt die Logik zur Authentifizierung. Es ist in Betracht zu ziehen, aus schließlich Programmcode auszuliefern, der clientseitig ausgeführt wird.

Die Interoperabilität wird durch eine definierte Methoden-Signatur gewährleistet. Die Parameter der Methode beziehen sich auf die Klasse `Endpoint` (siehe Abbildung 16) sowie den Namen und das Passwort eines Nutzers. Der Rückgabewert ist bei einem Erfolg ein User-Token der Dienstleistung.

Die Abbildung 21 beschreibt die Bypass-Authentifizierung in Prozessschritten.

![Abbildung 21: Bypass Authentication (Eigene Darstellung)](image)

Wird eine individualisierte Route angefragt, könnte ein Script direkt von einem Server des Middleware Consumer bezogen werden.

### 4.3.5 Das Aufsuchen und Beschreiben von Third Party Modules

Jede MWA kann mehrere Third Party Modules (TPM) beinhalten. Im Setup werden alle benötigten ausgewählt. Aufgrund der losen Kopplung zwischen MWAs und Hub hat dieser keine Informationen über die verfügbaren TPMs einer MWA. Diese Problematik erfordert einen Lösungsansatz, bei dem die MWA-Schnittstellen zum Abfragen von
TPM-Attributen bereitstellt. Im Folgenden werden die Attribute (siehe Abbildung 16 [S.43]) definiert.

Die Attribute `thirdparty` und `tpVersion` geben die zu adaptierende Anwendung mit der entsprechenden Versionsnummer an. Das Attribut `type` beschreibt, welcher MWA-Domäne das TPM angehört. Bei dem Attribut `description` wird das TPM beschrieben. Es können Hinweise zu Authentifizierungsmechanismen sowie zur Route-Konfiguration gegeben werden. Das Attribut `author` benennt den Entwickler des TPM.

Die TPM-Attribute haben eine hohe Relevanz für die Konfiguration von Routes. Technisch werden bei einem `lookup` alle TPMs gleichbehandelt. Eine fehlerhafte Konfiguration führt nicht zu Systemfehlern. Es besteht jedoch die Möglichkeit, dass ein TPM einer anderen als der vorgesehenen Domäne zugewiesen werden kann. Eine Clientanwendung würde davon ausgehen, dass die Route für die vorgesehene MWA-Domäne konfiguriert wurde. Die Folge wäre, dass keine Modulimplementierung in der MWA gefunden wird.

In der Abbildung 22 [S.50] versucht die Clientanwendung, mit einer Route der Domäne D1 einen Service einer MWA derselben Domäne anzufragen. Die Anfrage ist in diesem Fall erfolgreich. Eine weitere Anfrage mit der Route an eine MWA der Domäne D2 führt dazu, dass die Implementierung nicht gefunden wird.

Abbildung 22: Route Domain (Eigene Darstellung)
4.4 Der DAO Manager als Service Broker


Ein DAO besitzt eine eindeutige Lookup-ID. Diese setzt sich aus den nachfolgenden Attributen zusammen:

- Third Party [Module]
- Third Party Version [Module]
- Entity [API]
- ConnectorType [Attribut aus den CC]

Bei dem Entity handelt es sich um das standardisierte domänenspezifische Datenmodell der MWA.

Das Aufrufen der passenden CRUD-Methode erfolgt über die übermittelte Action der MWAs. Diese orientieren sich an dem REST-Paradigma (Fielding, 2000). Daraus folgen die entsprechenden Mappings:

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>DAO (CRUD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>Create-Methode</td>
</tr>
<tr>
<td>GET</td>
<td>Read-Methode</td>
</tr>
<tr>
<td>PUT</td>
<td>Update-Methode</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete-Methode</td>
</tr>
</tbody>
</table>

Der DAO Manager übermittelt neben dem Entity auch die Route und optionale Parameter. Auf diese Weise gelangt eine DAO-Methode über die Route an die UC und CC sowie die Endpunkt-Adresse. Aufgrund der injection aller nötigen Parameter kann ein DAO zustandslos initialisiert werden.
4.5 Skalierung und Load Balancing

Durch die lose Kopplung der Hauptkomponenten und dem zustandslosen Agieren der TPMs ist die gesamte MI skalierbar. Wie beschrieben, lassen sich mehrere TPMs einer MWA zuordnen.


Ein wesentlicher Nachteil der aktuellen Hub-Architektur ist der Bottleneck zwischen einem Hub und mehreren MWAs. Bei jeder Clientanfrage fordert eine MWA die Informationen einer Route an. Hierbei entsteht zudem ein unnötiger Overhead.


Ein Hub kann mit dem Entwurfsmuster Publish & Subscribe alle registrierten MWAs über Veränderungen von Routes oder Middleware Consumers benachrichtigen. Die MWAs reagieren auf die Benachrichtigung mit einer Invalidierung des lokalen Zwischenspeichers. Bei einer erneuten Clientanfrage fordert eine MWA die veränderte Route von dem Hub an (engl. fetching).


Der Proxy besitzt zwei Balancer Cluster C1 und C2. Der Cluster C1 wird allen MWAs der Domäne D1 zugeordnet. C2 enthält nur Balancer Member der Domäne D2. Der Hub 1 nimmt Authentifizierungsanfragen von den MWAs der Domäne D1 an. Der Hub 2 nimmt Anfragen aus der Domäne D1 und D2 an. Aus der Domäne D1 wird nur die zweite MWA-Instanz akzeptiert.
4.6 Die Integration von Drittanbietern aus der E-Learning-Branche

In diesem Kapitel wird für den bestehenden Systementwurf beschrieben, wie Dienstleistungen der E-Learning-Branche integriert werden könnten.

In dem Kapitel Systemanalyse wurden Annahmen getroffen, wie Dienstleistungen der Branche in Microservices unterteilt werden könnten. Es ist eine große Herausforderung, Dienstleistungen in eigenständige Komponenten zu bündeln, sodass keine unnötigen Abhängigkeiten entstehen.

Zusammenfassend sind die Dienstleistungskomponenten der E-Learning-Branche, die in dieser Arbeit verwendet werden, im Folgenden aufgelistet:

- Tracking (Learning Record Store)
- Learning Management System (Course)
- Communication (Forums)


<table>
<thead>
<tr>
<th>API</th>
<th>Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracking (LRS)</td>
<td>ExperienceAPI (xAPI) (TinCan, n.d.)</td>
</tr>
<tr>
<td>LMS</td>
<td>IMS CC (Course) (IMS Global Learning Consortium, 2015a)</td>
</tr>
</tbody>
</table>

Das Ergebnis ist eine Clientanwendung, die mit marktführenden oder etablierten Anwendungen kompatibel ist. Mehrere Bildungseinrichtungen können gemeinsam eine Clientanwendung entwickeln. Erst durch das Corporate Design und die jeweiligen Routes der Bildungseinrichtungen wird die Clientanwendung individualisiert.

Aufgrund von Projektanforderungen und der Erreichbarkeit von LMS wurden die Anwendungen Moodle 2, Moodle 3 und OpenOLAT als TPMs integriert. Für das Tracking ist vorerst nur die Anwendung Learning Locker vorgesehen.


Abbildung 24: Middleware Infrastructure - E-Learning (Eigene Darstellung)

Durch die Orchestration der MWAs LRS, LMS und COM ist es möglich, Clientanwendungen, wie beispielsweise eine Lernbegleiter-App, ein Management-System oder ein Authoring Tool zu implementieren.
4.7 Zusammenfassung

Zusammenfassend wurde bei dem Systementwurf ein zentraler Hub integriert, der sowohl Service Registry als auch Authentifizierungsstelle ist. Es gibt keine Kommunikation zwischen Drittanbietern (Service Provider), wie es bei den SOAs Hub & Spoke und Enterprise Service Bus (ESB) der Fall ist (n:n-Relation).


Jede MWA wird domänenspezifisch ausgelegt und kann mehrere TPMs für Drittanbieteranwendungen bereitstellen. Basierend auf dem domänenspezifischen Ansatz wurden die MWAs COM (Kommunikation), LRS (Aufzeichnen von Lernaktivitäten) und LMS (Kurs-, Nutzer- und Lerninhalte) für die E-Learning-Branche in die MI überführt.
5 Implementierung

In diesem Kapitel wird basierend auf dem Systementwurf eine logische Codestruktur für die MI erörtert. Es werden die Kernfunktionalitäten in Form von funktionalen Modulen vorgestellt.

Ein funktionales Modul ist ein Versuch, auf Codeebene eine lose Kopplung von Funktionalitäten zu realisieren. Relevante Codefragmente eines funktionalen Moduls werden mit Codeausschnitten belegt.

Des Weiteren werden die E-Learning MWAs COM, LRS und LMS betrachtet. Hierbei sind die bisher ermittelten domänenspezifischen Standards zu beschreiben.

Abschließend erfolgt eine Auflistung und Kategorisierung der benötigten Hub-Services.

5.1 Die logische Codestruktur der Middleware Infrastructure

Eine logische Codestruktur soll eine klare Trennung von Entwickleraufgaben sicherstellen. Es ist zu vermeiden, dass ein Entwickler Programmcode verstehen oder bearbeiten muss, der für die Lösung des Problems irrelevant ist.

Dieses Kapitel beschreibt die im Rahmen dieser Arbeit eingeführten funktionalen Module der MI. Es besteht kein Zusammenhang zu den TPMs der MI, die für die Adaptierung von Drittanbieteranwendungen verantwortlich sind.

Die funktionalen Module kapseln Kernfunktionalitäten der MI. Fast alle funktionalen Module definieren Schnittstellen oder Annotationen für Entwickler, um die Komplexität zu verbergen. Hierbei wird das Umsetzungsparadigma Inversion of Control (IoC) berücksichtigt (Fowler, 2004).


Innerhalb eines TPM werden alle Abhängigkeiten (engl. *dependencies*) über das Build-Management-Tool *Maven* mit dem Scope *provided* versehen. Das bedeutet, dass nur die MWA, die im späteren Verlauf TPMs einbindet, Abhängigkeiten zur Verfügung stellt.

Die folgenden Kapitel betrachten die funktionalen Module im Detail. Schnittstellen sowie Annotationen werden erläutert und relevante Kernfunktionen durch Codeausschnitte belegt.

### 5.1.1 Die Bereitstellung von Data Access Objects (DAO)

Das Modul *DAO* regelt den Kontrollfluss zwischen der MWA und einem TPM. Dieses kann verschiedene Schnittstellen zu Funktionen einer *Drittanbieteranwendung* bereitstellen.


Es wird das AbstractRouteDAO Interface bereitgestellt. Neben den CRUD-Methoden sind einige Parameter vorgegeben, die von der MI bei der Initialisierung übergeben werden. Das Listing 1 [S.58] zeigt das Codegerüst eines DAO.
public abstract class AbstractRouteDAO<E> {
    public abstract E create(Route route, E entity, Map<String, Object> optionalParams);
    public abstract List<E> read(Route route, E entity, Map<String, Object> optionalParams);
    public abstract E update(Route route, E entity, Map<String, Object> optionalParams);
    public abstract boolean delete(Route route, E entity, Map<String, Object> optionalParams);
}

Listing 1: CRUD Methods - AbstractRouteDAO (Eigenes Listing)

Jede Methodensignatur beinhaltet in dem Listing 1 [S.58] drei Parameter:

Der erste Parameter enthält das Objekt Route. Durch die Route kann ein DAO Informationen für einen Verbindungsaufbau zu einem Endpunkt erlangen.

Der zweite Parameter beinhaltet das generische Datenobjekt (Entity), das einen Standard aus der MWA abbildet. Die Klasse CourseDAO würde beispielsweise mit dem Entity Course typisiert werden.


Wie bereits im Systementwurf angedeutet, wird ein business delegate verwendet, um bei einer Clientanfrage an die MWA das benötigte DAO zu initialisieren. Aufgrund der Java-EE-Umgebung wird ein AbstractRouteDAO über das Java Naming and Directory Interface (JNDI, Oracle, n.d.) identifiziert.


An dieser Stelle kommt die Klasse `DAOManager` zum Einsatz. Anstatt die `Context and Dependency Injection` (CDI, Oracle, 2013a) Annotation `@EJB` zu verwenden, wird die Logik zum Aufruf eines EJB durch den `DAOManager` implementiert.

Das Listing 2 [S.59] zeigt, wie eine zum Zeitpunkt der Clientanfrage erzeugte dynamische JNDI dazu genutzt werden kann, ein DAO als EJB zu initialisieren und aufzurufen.

```java
public <T> T lookup(String jndi) {
    Context context;
    try {
        context = new InitialContext();
        Object obj = context.lookup(jndi);
        return (T) obj;
    } catch (NamingException e) {
        logger.error("EJB not found for JNDI -> " + jndi);
        throw new CustomException("EJB not found","500",this.getClass());
    }
}
```

Listing 2: DAO lookup method - DAOManager (Eigenes Listing)

In der Zeile 4 wird ein neuer `InitialContext` instanziert. Dieser implementiert das `Context` Interface und stellt einen Kontext zum Auflösen einer JNDI her. Die Methode `lookup` in Zeile 5 liefert entsprechend der übermittelten JNDI ein EJB. Im Falle, dass unter der JNDI kein EJB zu finden ist, wird eine entsprechende `CustomException` geworfen und ein Logeintrag vermerkt.

Der `lookup` wird in dem entsprechenden MWA-Service ausgelöst. Dazu wird vorerst der `DAOManager` als `Singleton` (Gamma et al., 2011, pp. 157-170) über CDI eingebunden. Dieser besitzt eine Methode, um die JNDI zu erstellen (siehe Listing 3 [S.59]).

```java
public static final String getJNDI(final Route route, final Class<?> entity) {
    String thirdparty = route.getModule().getThirdparty();
    String version = route.getModule().getTpVersion();
    Security security = route.getSecurity();
    ConsumerCredentials cc = security.getConsumerCredentials();
    ConnectorType connectorType = cc.getConnectorType();

    return JNDI_MODULE_PREFIX + thirdparty + "_" + version + "_" + entity.getSimpleName() + "_" + connectorType.name();
}
```

Listing 3: Dynamic JNDI (Eigenes Listing)

In den Zeilen 2 bis 6 werden die Attribute der Route nicht auf Nullpointer überprüft, was normalerweise nicht zu empfehlen ist. Routes werden in verschiedenen Programmteilen verwendet, unter anderem auch bei der Entwicklung eines TPM. Aus diesem Grund wird in dem funktionalen Modul Routing sichergestellt, dass alle Pflichtattribute vorhanden sind. Der vorliegende Programmcode wird erst nach der Datenvalidierung aufgerufen.

Die Pflichtattribute wurden bereits in dem Systementwurf beschrieben und werden an dieser Stelle nur in einem praktischen Beispiel verdeutlicht:

**Beispiel:** Gesucht wird ein DAO, das Schnittstellen für einen Kurs bereitstellt. Die Schnittstellen des DAO sollen ein Moodle-System mit der Version 3.0 über Webservices abrufen. Die JNDI würde entsprechend der Vorgaben nach Aufruf der Methode aus dem Listing 3 wie folgt an die Methode lookup übergeben werden.

```
JNDI: java:module/moodle_3.0_Course_HTTP
```

Das CourseDAO ist mit der @Stateless(value=JNDI) zu annotieren. Das Listing 4 [S.60] zeigt einen Teil der Logik aus dem CourseService der LMS MWA. In der Zeile 7 wird die JNDI entsprechend dem Entity (Course) und der Route generiert. Durch die Zeile 10 wird ein DAO ermittelt.

Besitzt ein TPM das CourseDAO, wird dieses im zweiten Parameter als Delegate Objekt übergeben. Der dritte Parameter gibt die auszuführende CRUD-Operation an.

Aufgerufen wird in Zeile 12 des Listing 4 [S.60] die Methode read von dem CourseDAO. An dieser Stelle besteht die Möglichkeit, die Route im ersten Parameter und die optionalen Parameter im fünften Parameter zu übergeben. Bei Bedarf könnte auch ein Entity (Course) im vierten Parameter übermittelt werden.

```
@EJB
private DAOManager daoManager;

public ContainerResponse<Course> getCourses(/∗ API Params */) {
    // create dynamic JNDI based on Course entity and Route
    String jndi = DAOManager.getJNDI(route , Course . class ) ;

    // find a potential DAO with given JNDI
    AbstractRouteDAO<Course> courseDAO = daoManager.lookup(jndi);

    // call specified READ method on courseDAO delegate
    return daoManager.callMethod(route , courseDAO , Action.READ , null ,
                                   optParams);
}
```

Listing 4: Find CourseDAO (Eigenes Listing)
Neben dem `AbstractRouteDAO` und dem `DAOManager` werden zwei Filter-Annotationen für ein DAO bereitgestellt. Diese eignen sich, um boilerplate code zu vermeiden. Der Begriff beschreibt eine Problematik in der Informatik, bei der bestimmte Codefragmente in nahezu unveränderter Form an verschiedenen Stellen erneut eingesetzt werden.

Die Filter, angelehnt an Alur et al. (2001, pp. 144-163), sind in der Lage, die besagten Codefragmente auszulagern und über IoC vor (`@PreFilters`) oder nach (`@PostFilters`) dem Ausführen einer CRUD-Operation aufzurufen.


```java
public interface Filter {
    public void filter(Route route, Object entity, Map<String, Object> optionalParams, Action action);
}

@LocalBean
@Stateless(name = JNDI.COURSE_HTTP_DAO)
public class CourseDAO extends AbstractRouteDAO<Course> {
    @Override
    @PreFilters(SecurityFilter.class, Filter2.class)
    @PostFilters(Filter3.class, Filter4.class)
    public List<Course> read(Route route, Course course, Map<String, Object> optionalParams) {
        // some code ...
    }
}
```

Listing 5: Filter and DAO registration (Eigenes Listing)

Die `Filter Chain` aus dem Listing 5 [S.61] legt folgende Reihenfolge fest:

SecurityFilter → Filter2 → read() → Filter3 → Filter4

Der `SecurityFilter` in Zeile 10 könnte eine für das TPM geeignete Authentifizierungsstrategie implementieren. UC und CC würden vor dem Aufruf der Methode `read` aus der `Route` ausgelesen und validiert werden.

In den Zeilen 5 und 6 wird zudem das `CourseDAO` als `Stateless` EJB registriert. Die Konstante `JNDI.COURSE_HTTP_DAO` beschreibt entsprechend der Vorgabe die JNDI, unter der das `CourseDAO` zu finden ist. Die Annotation `@LocalBean` gibt an, dass es sich um ein EJB handelt, welches kein Interface besitzt. Das EJB wird zudem lokal in der gleichen Applikationsserver-Instanz von dem `DAOManager` verwendet.
5.1.2 Die Bereitstellung von Data Transfer Objects (DTO)


```
{
  "data": null,
  "status": {
    "code": "OK",
    "msg": "200"
  }
}
```

**Listing 6: ContainerResponse - JSON (Eigenes Listing)**

Das DTO `ContainerResponse` hat zwei hilfreiche Funktionen. Zum einen wird ein einheitliches Containerformat für alle Services der MWA ermöglicht und zum anderen ist neben dem Status aus dem HTTP-Protokoll ein weiteres `Status`-Objekt enthalten.

Durch das Objekt `Status` können Fehlermeldungen direkt von einer *Drittanbieteranwendung* über die MWA an die *Clientanwendung* weitergeleitet werden.

Für eine *Clientanwendung* ist es von Vorteil zu erkennen, ob ein Fehler bei der Verbindung zwischen der MWA oder der *Drittanbieteranwendung* aufgetreten ist. Entstehen Verbindungsprobleme zwischen der *Clientanwendung* und der MWA, wird der Status aus dem HTTP-Protokoll verwendet.


```
ContainerResponse.ok().entity(response).build();
```

**Listing 7: ContainerResponse - Builder Pattern (Eigenes Listing)**

Ein Record besitzt die Attribute id, consumerID, consumerName, timestamp, thirdparty, tpVersion, serviceType, connectorType, routeID und routeDisplayName.

Der Hub bietet unter anderem eine REST-Schnittstelle, um die Records über das Management Interface einsehen zu können.

5.1.3 Die Fehlerbehandlung der Middleware Infrastructure (Exception)

Grundsätzlich ist eine saubere Fehlerbehandlung für einen robusten Programmcode sehr wichtig. In Java kann eine Fehlerbehandlung sehr mühsam sein, besonders, wenn sehr viele verschiedene Anwendungsfälle durch Clientanfragen eintreten könnten.

Aus diesem Grund wird für jede MWA ein standardmäßiger ExceptionMapper vorgesehen. Fehlermeldungen, die nicht gesondert zu behandeln oder während der Entwicklung unbemerkt geblieben sind, werden nicht als stacktrace an die Clientanwendung gesendet. Zum einen ist ein stacktrace für die Clientanwendung nicht geeignet und zum anderen könnten systeminterne Daten veröffentlicht werden. Dies kann unter Umständen ein Sicherheitsrisiko bedeuten.

Alle nicht behandelten Exceptions werden durch das DTO ContainerResponse aus dem funktionalen Modul DTO in menschen- und maschinenlesbarer Form an die Clientanwendung übermittelt.

Neben dem ExceptionMapper wird eine CustomException zur Verfügung gestellt. Diese kann bewusst eingesetzt werden, um zu signalisieren, dass der aufgetretene Fehler nicht zu behandeln ist.

Bei der CustomException handelt es sich um eine RuntimeException.

Der ExceptionMapper vermerkt in diesem Fall die entwicklereigene Fehlermeldung im Server Log für andere Entwickler und Systemadministratoren. Zudem wird die Clientanwendung auf den Fehler hingewiesen.

Ein typischer Fehler könnte die Missachtung von Attributen sein, die für ein bestimmtes TPM benötigt werden.

5.1.4 Die Integration von DAO-Verbindungstypen (Connector)

Jeder Entwickler kann frei entscheiden, welcher Verbindungstyp für ein DAO verwendet werden soll. Wichtig ist, dass der ConnectorType in der JNDI dem verwendeten Connector des DAO entspricht.
Es gibt aktuell kein einheitliches Interface für einen Connector, da Verbindungspro-
tokolle sehr verschieden sein können. Der erste und bisher einzige Connector ist der HTTPConnector. Dieser wird zum Aufrufen von Webservices (vorzugsweise REST) ge-
nutzt.

Die Implementierung des HTTPConnector wird durch das RestEasy Client Proxy Fram-
ework realisiert. Die verwendete Bibliothek RestEasy basiert auf der JAX-RS Client API
2.0 Spezifikation (Oracle, 2013b).

5.1.5 Die Konfiguration von Hub und Middleware API (Config)

Hinsichtlich der Wartbarkeit einer MWA und eines Hubs soll verhindert werden, dass Konfigurationsparameter über den Programmcode zu verändern sind.

Aus diesem Grund wird eine Properties-Datei zur Verfügung gestellt. Als Vorausset-
zung gilt, dass die Datei über die Serverumgebung erreichbar ist. Es wird nach der
Datei slhw.properties gesucht.

An dieser Stelle wird auf eine Implementierung von (Ritchie, 2015) verwiesen, in der eine Properties-Datei zum Start eines Deployments ausgelesen wird.

Anschließend ist es möglich, die gewünschten Properties in EJB-Variable mit der
@Inject und der erzeugten Annotation @ApplicationProperty(propertyName) zu
injizieren, wie das Listing 8 [S.64] verdeutlicht.

```java
@Inject
@ApplicationProperty(name = "mw_apiKey")
private String apiKey;
```

Listing 8: Property injection (Eigenes Listing)

5.1.6 Das Management von Routen (Routing)

Das funktionale Modul DAO stellt unter anderem mit den Informationen aus den Routes in der MWA ein geeignetes DAO zur Verfügung.

Wie eine Route aus dem Hub nach Anfrage der Clientanwendung in die MWA gelangt,
ist bisher auf Ebene der Implementierung nicht erläutert worden. Für diese Aufgabe
ist das funktionale Modul Routing verantwortlich.

Eine elementare Klasse für die nachfolgenden Prozesse ist der HubManager. Dieser wird,
wie auch der DAOManager in den MWA-Services, als Singleton eingebunden. Der Hub-
Manager fragt eine Route mithilfe der von der Clientanwendung übermittelten RouteID von dem Hub an.

Bei dem Erhalt einer Route werden die Daten auf Vollständigkeit überprüft und validiert. Dieser Prozessschritt wurde bereits im Zusammenhang mit dem Überprüfen von Nullpointer auf Route-Attributen in dem funktionalen Modul DAO erwähnt.


```java
@Lock(LockType.WRITE)
public void addRoute(String hubURL, String consumerToken, Route route) {
    String host = getHost(hubURL);
    if (!hubStorage.containsKey(host)) {
        hubStorage.put(host, new HashMap<String, Route>());
    }

    String key = getKey(consumerToken, route.getId());
    hubStorage.get(host).put(key, route);
}

public String getKey(String consumerToken, String routeID) {
    return Base64.encodeBytes((consumerToken + "":" + routeID).
    getBytes());
}
```

Listing 9: Add Route to HubStorage (Eigenes Listing)

In der Zeile 4 wird überprüft, ob für den aktuellen Hub, von dem eine Route angefragt wurde, bereits eine HashMap vorliegt. Ist dies nicht der Fall, wird eine neue HashMap initialisiert.

Für die angefragte Route wird auf Basis des consumerTokens und der routeID in der Zeile 8 ein Key erstellt. Der Key ist eine Base64-Konkatenation der beiden Parameter, wie die Zeile 13 verdeutlicht.

Die Zeile 9 fügt die Route mit dem erstellten Key der HashMap (Zwischenspeicher) hinzu.

Wird zu einem späteren Zeitpunkt die Route erneut benötigt, kann vor einer Anfrage anhand des Keys überprüft werden, ob die Route bereits lokal vorliegt. In diesem Fall wird der HubManager keine weitere Anfrage über den HubClient an den Hub stellen.
Wenn die lokal vorliegenden Routes nicht mit den Routes des Hubs übereinstimmen, kann der Hub die MWA veranlassen, die Daten der entsprechenden HashMap zu entfernen. Daraus folgt, dass eine benötigte Route von dem Hub angefragt und anschließend erneut lokal in eine HashMap geschrieben wird.

Das Zwischenspeichern ist nötig, um zu gewährleisten, dass der zentrale Hub nicht durch zu viele Anfragen überlastet wird.

Der HubClient authentifiziert die MWA gegenüber dem Hub. Die MWA wird autorisiert, Routes anfragen zu können. Der benötigte Middleware Key wird über eine Property aus dem funktionalen Modul Config bereitgestellt.

5.1.7 Das Abfragen von verfügbaren Modulen (Scan)

Eine MWA liefert einem Hub oder anderen externen Schnittstellen Informationen zu verfügbaren TPMs. Zu diesem Zweck wird ein ScanService durch das funktionale Modul Scan bereitgestellt. Ein TPM ist verfügbar, wenn dieses durch Maven in der MWA eingebunden wurde.

Das Listing 10 [S.66] zeigt, wie über den ClassLoader einer MWA Ressourcen aufgesucht werden können.

```java
public ContainerResponse<?> scan() {
    ClassLoader cl = Thread.currentThread().getContextClassLoader();
    try {
        // searching for module.properties files
        Enumeration<URL> urls = cl.getResources(RELECTION_MARKER);
        while (urls.hasMoreElements()) {
            URL url = (URL) urls.nextElement();
            // create virtual file from resource url
            VirtualFile vf = asVirtualFile(url);
            // get required physical path
            File file = vf.getAbsolutePath();
            // read module properties from physical file
            readModule(file);
        }
    } catch (IOException e) { /* exception handling */ } 
    // return all available module descriptions
    return ContainerResponse.ok().entity(modules).build();
}
```

Listing 10: Third Party Module (TPM) scan (Eigenes Listing)

Durch die Konstante REFLECTION_MARKER mit dem Wert META-INF/module.properties werden Dateien mit dem Namen module.properties ermittelt. Aus diesen Dateien werden die Beschreibungen zu TPMs ausgelesen und über den ScanService zur Verfügung gestellt.
An dieser Stelle ist eine Bedingung zu erwähnen, ohne die das Aufsuchen von TPMs nicht, wie in dem Listing 10 [S.66], realisiert werden kann.

Bei der Erstellung eines TPM muss ein Entwickler für außenstehende Dienste und Nutzer eine Modulbeschreibung ausliefern. Dazu wird in dem EJB-Projekt für das TPM in dem Ordner **META-INF** die Datei *module.properties* angelegt. Diese enthält die Modulbeschreibungen.


Bevor eine Modulbeschreibung über den *ScanService* vermittelt werden kann, muss eine physikalische Datei vorliegen. Der *Wildfly*-Server verfügt für die Verwaltung von *Deployments* über ein *read-only Virtual File System* (**VFS**, *JBoss*, 2008, chapter 20).


```
private VirtualFile asVirtualFile(URL url) {
    // check if the given url is type of virtual file
    if ("vfs".equals(url.getProtocol())) {
        try {
            URLConnection conn = url.openConnection();
            // extract virtual file from url content
            VirtualFile virtualFile = (VirtualFile)conn.getContentType();
            return virtualFile;
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

Listing 11: Virtual file to physical file (Eigenes Listing)

Sobald die physikalische Datei vorliegt, kann in Zeile 13 des Listing 10 [S.66] mithilfe der Klasse *java.util.Properties* jedes Attribut durch die Methode *readModule(File)* ausgelesen werden.
5.1.8 Die modulübergreifenden Funktionalitäten (Shared)

Einige Codebestandteile werden in vielen funktionalen Modulen intensiv eingesetzt, womit es schwierig ist, klare Schnittstellen zu schaffen. Zudem haben diese hinsichtlich der Nutzung eine allgemeine Charakteristik.

Bei den Datenmodellen handelt es sich um Route und Consumer. Diese werden von dem Hub, der MWA und den TPMs verwendet.

Im ursprünglichen Sinne war das funktionale Modul Shared ausschließlich für Hilfsklassen vorgesehen. Häufig benötigte Codefragmente sollen an einer Stelle zusammengefasst werden, um die Entwickler bei der Entwicklung von TPMs zu unterstützen. Daraus folgt, dass der Programmcode übersichtlicher und einheitlicher gestaltet werden kann. Es steht den Entwicklern jedoch frei, auch abweichende Implementierungen zu verwenden.

Das funktionale Modul enthält zudem einige Header-Konstanten, die bei der Implementierung von MWAs in dem folgenden Kapitel eine hohe Relevanz aufweisen.

5.1.9 Die Implementierung von Middleware APIs

Bei der Entwicklung einer MWA gibt es zwei Schwerpunkte. Die Konzeption der API-Schnittstellen, dazu gehört die Service URL sowie Path-, Query und Header-Parameter und das Definieren von Service-Standards.

Es gibt keine Vorgaben, wie die URL einer Schnittstelle auszusehen hat. Daher wird empfohlen, ein einheitliches Konzept für alle MWAs zu verwenden. Im Rahmen dieser Arbeit wurden REST-Services bereitgestellt.

Es existieren vier Pflicht-Header, die für das Routing und die Middleware Consumer Authentifizierung benötigt werden:

- Der Authorization Header übermittelt den Token des Middleware Consumers. Über diesen Token kann mithilfe des HubManager ermittelt werden, auf welche Routes der Consumer Zugriff hat.
- Der X-Hub Header gibt den Hub an, bei dem der HubManager Routes anfragen kann.
- Der X-Route-Id Header identifiziert die Route, welche von dem HubManager angefragt und von dem DAOManager verwendet werden soll.
- Der X-TP-Auth Header bietet die Möglichkeit, UserCredentials (UC) an die MWA zu übermitteln.
Für jede MWA wird ein getrenntes Model EJB-Modul angelegt, in dem alle domänenspezifischen Standards definiert werden. Das Model EJB-Modul wird von den TPMs und der MWA verwendet (siehe Abbildung 25 [S.57]).


Nachfolgend werden die Standards von COM, LRS und LMS betrachtet. Hierbei handelt es sich um die MWAs, die im Rahmen dieser Arbeit realisiert werden.

Die COM API


Die LRS API

Die LRS API verwendet die experienceAPI (xAPI) und ist für das Aufzeichnen von Lernaktivitäten verantwortlich.

Bei dem Aufzeichnen (Tracking) besteht das Problem, dass für die Analytics so viele Daten wie möglich eingeholt werden sollten. Dies führt zu domänenspezifischen Attributen.

Ein Vorteil des xAPI-Standards ist die Verwendung von Internationalized Resource Identifiern (IRI). Hierbei wird die Methodik des linked data begünstigt, bei der die konkreten Ressourcen oder Metadaten maschinenlesbar referenziert werden.
Bei dem xAPI-Standard, auch TinCan API genannt, basiert ein Record beziehungsweise Statement auf der Aussage „I did this“ (Nomen, Verb, Object). Hierbei ist das Nomen der Akteur einer Lernaktivität, das Verb die eigentliche Tätigkeit und das Object der Lerninhalt. (vgl. TinCan, n.d.)

Ein Statement besitzt neben dem Akteur, der Tätigkeit und der Lernaktivität noch weitere Attribute. Aktuell wird xAPI aufgrund der logischen und flexiblen Struktur verwendet.

Im Rahmen dieser Arbeit sind die Grundeigenschaften und die damit verbundenen Möglichkeiten von Interesse. Statt jedes Attribut des Standards im Detail zu beschreiben, erfolgt an dieser Stelle eine Auflistung von Grundfragen, die ein Statement in Form vonAttributen beantworten sollte:

- Wer ist der Akteur?
- Welche Aktion wird getätigt?
- Auf welchen Inhalt bezieht sich die Aktion?
- Zu welchem Zeitpunkt wurde die Aktion ausgeführt?
- Wie lange wurde die Tätigkeit ausgeübt?
- Mit welchem Ergebnis wurde die Tätigkeit beendet?
- Gibt es eine Aktion, die von einer anderen Aktion abhängig ist?

Langfristig ist geplant, viele Attribute in der LRS API durch eine IRI zu ersetzen. Die verschiedenen MWAs nach dem REST-Paradigma liefern eine gute Grundlage, um konkrete Informationen oder Metadaten mithilfe einer IRI bereitzustellen.

Auf diese Weise können minimalistiche Statements durch referenzierte domänenspezifische Datenmodelle in beliebiger Tiefe angereichert werden. Da eine MWA Standards einer Domäne abdeckt, ist es einer Clientanwendung möglich, die Informationen maschinell verarbeiten zu können.

Durch das Routing- und Consumer-Management eines Hubs könnte die Integration des linked data dazu führen, dass ein ausgeliefertes Datenmodell bestimmte Referenzen nicht für alle Consumer veröffentlicht. Die Integration von linked data ist somit nicht ausschließlich für die LRS-Schnittstelle von Interesse.

Die Abbildung 27 [S.71] verdeutlicht die aktuelle Integration der LRS MWA. Es werden die Attribute, die referenzieller Natur sind, angedeutet. Im Anhang A befindet sich ein entsprechendes Statement.
Sowohl die Verben als auch die Objekte werden über eine IRI referenziert. *ADLNet Vocabulary* ist eine offizielle Datenbank mit maschinenlesbaren Beschreibungen zu Verben.

Das LCMS-Repository ist eine beliebige Ablage von Lerninhalten inklusive der Metadaten. An dieser Stelle wäre die Integration einer LCMS MWA sinnvoll.

Das Objekt *Result* besitzt einige Attribute, die nicht zu referenzieren sind, dazu gehören Duration, Success und Completion. Für die Attribute *Response* und *Score* wäre es nach *linked data* interessant, eine weitere MWA einzubinden, da die benannten Attribute sehr spezifisch sein können.

Auch der Akteur könnte theoretisch statt des *mBox* Attributes (E-Mail Adresse) durch eine USR API identifiziert werden.

**Die LMS API**


Die Abbildung 28 [S.72] zeigt die Ressourcentypen, die durch den IMS-CC-Standard berücksichtigt werden.

Abbildung 28: Part of IMS CC specification (IMS Global Learning Consortium, 2015a)
Im Rahmen dieser Arbeit werden nur externe Files (URL) und auf LTI basierende Lernobjekte erfasst. Discussion Topics werden durch den COM-Service extern bereitgestellt.

LTI-Lernobjekte sind Bestandteil der LMS MWA und werden über einen LTI-Service zur Verfügung gestellt.

Es ist zu erwähnen, dass der tatsächliche Inhalt eines Lernobjekts über ein content-Attribut in Form eines `stringify` JSON ausgeliefert wird. Auch Quizaufgaben nach IMS QTI Spezifikation (IMS Global Learning Consortium, 2015b) werden auf diese Weise behandelt. Aus diesem Ansatz ergibt sich eine lose Kopplung zwischen Lerninhalt und LMS.

Das Tracking kann optional, feingranular und inhaltspezifisch über die LRS MWA in einem LTI-Lernobjekt erfolgen. Ein JSON des abgewandelten IMS CC befindet sich im Anhang B.

### 5.2 Die Schnittstellen der Hub API

Der *Hub* ist für die Verwaltung von *Routes* und *Consumers* verantwortlich. Im Detail ist der *Hub* ein klassisches *Backend* und stellt REST-Services für die Entities *Route*, *Consumer*, *Middleware*, *Endpoint*, *Config* und *Record* bereit.

Der *Hub* besitzt eine Management-, Route- und Public-API, wie die Tabelle 7 auf S.73 aufzeigt:

<table>
<thead>
<tr>
<th>Services</th>
<th>Management API</th>
<th>Route API</th>
<th>Public API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auth</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Config</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Route</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Middleware</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Endpoint</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Middleware Notification</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Tracking</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Consumer</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RouteInfo</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

**Multipage Management Interface** entwickelt. Im Anhang C befindet sich eine Ansicht zum Editieren eines *Middleware Consumer*. Die Management-API ist ausschließlich mit einem gültigen Management Key zu nutzen.

Die Route-API wird von dem **HubManager** verwendet, um gezielt eine *Route* für eine MWA anzufragen. In dieser *Route* sind alle Verbindungsparameter und Credentials enthalten. Die Route API kann ausschließlich mit einem gültigen Middleware Key genutzt werden. Es wird vorausgesetzt, dass der *Hub* die MWA mit entsprechendem Key registriert hat.


### 5.3 Zusammenfassung

Zusammenfassend soll eine logische Codestruktur eine klare Trennung von Entwickleraufgaben sicherstellen.


Die MI besteht aus den in Tabelle 8 [S.74] abgebildeten funktionalen Modulen:

<table>
<thead>
<tr>
<th>Funktionales Modul</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan</td>
<td>Auffüllen von verfügbaren Third Party Modules in der Middleware API</td>
</tr>
<tr>
<td>DAO</td>
<td>Aufsuchen von Drittanbieterschnittstellen nach CRUD-Operationen</td>
</tr>
<tr>
<td>DTO</td>
<td>Datentransferobjekte für einen einheitlichen Datenaustausch</td>
</tr>
<tr>
<td>Exception</td>
<td>zentrale Fehlerbehandlung in der Middleware Infrastructure</td>
</tr>
<tr>
<td>Shared</td>
<td>Util-Klassen und zwischen funktionalen Modulen geteilte Datenmodelle</td>
</tr>
<tr>
<td>Connector</td>
<td>Verbindungstypen zu Drittanbieteranwendungen für Third Party Modules</td>
</tr>
<tr>
<td>Config</td>
<td>Einlesen einer Konfigurationsdatei für die Middleware Infrastructure</td>
</tr>
<tr>
<td>Routing</td>
<td>Anfragen von Routen</td>
</tr>
</tbody>
</table>
Eine MWA wird als *Microservice* in Form eines *Web Application Archive* (WAR) veröffentlicht. Jede MWA verwendet ein zusätzliches EJB-Projekt für domänenspezifische Standards (Model).

Es wurden die Standards IMS CC (LMS API) und xAPI (LRS API) vorgestellt. Ein Third Party Module orientiert sich bei der Adaptierung ausschließlich an einem Model.

Der *Hub* kategorisiert externe Zugriffe nach der Public-, Management- und Route-API.

6 Evaluation


6.1 Qualitative Evaluation

Bei der qualitativen Evaluation steht die Bewertung der Integrierbarkeit und Nutzung im Vordergrund. In der Regel erfolgt in diesem Zusammenhang eine Nutzerumfrage. Im Rahmen dieser Arbeit wurde keine Nutzerumfrage erstellt, da die vollständige Integration der MI in die Systemumgebung des SLHw-Projekts zu diesem Zeitpunkt nicht abgeschlossen ist. Aus diesem Grund ist eine umfangreiche Entwicklerumfrage erst nach der Integration möglich.

In der vorliegenden Arbeit wird der Ist-Zustand der entwickelten MI betrachtet. Der Einsatz der MI in dem SLHw-Projekt ermöglichte die Bereitstellung von Kursen an den Institutionen Beuth Hochschule für Technik Berlin, Fraunhofer Institut für offene Kommunikation (FOKUS) und der Handwerkskammer Berlin.

Evaluierter werden die Kernziele dieser Arbeit. Zu diesen gehören die Interoperabilität, die Dynamisierung von Verbindungswegen (Routing), die Datenhoheit der Nutzer, die Authentifizierung von Clientanwendungen und die branchenübergreifenden Einsatzmöglichkeiten.

Zu Beginn werden die Kurse der jeweiligen Institutionen vorgestellt, gefolgt von einer Beschreibung der Integration der MI in die Systemumgebung des SLHw-Projekts. Abschließend erfolgt eine konkrete Evaluation der Kernziele.

6.1.1 Anwendungsfälle (Kurse)

In diesem Abschnitt werden reale Kurse vorgestellt, die in dem SLHw-Projekt über die MI angeboten wurden.
Gebäudeenergieberater Kurs (Projekt Smart Learning im Handwerk)


**Advanced Web Technologies Kurs (TU Berlin / Fraunhofer FOKUS)**


**JavaFX Kurs (Beuth Hochschule für Technik Berlin)**


6.1.2 Die Produktumgebung (Integration)

Die digitale Infrastruktur des SLHw-Projekts, auch openLCMSI genannt, verfolgt das Ziel, LMS- und LCMS-Inhalte als Dienstleistungen anzubieten.

Ein besonderer Fokus besteht darin, eine wiederverwendbare und vor allem generische Infrastruktur für verschiedene Nutzer mit unterschiedlichen *Clientanwendungen* bereitzustellen. Verschiedene Kurse sollen von unterschiedlichen Institutionen angeboten werden, die auch andere Themen behandeln können. (Krauss et al., 2016b)

Neben der Bereitstellung von Kursinhalten wird ein *Co-Design* zwischen der digitalen Infrastruktur und *Learning Analytics* (LA) angestrebt. Das Ziel ist es, für jeden Lerninhalt zusätzlich die Benutzerinteraktionen feingranular aufzeichnen zu können. Auf diese Weise wird es ermöglicht, in erster Linie dem Dozenten eines Kurses eine Über-
sicht über das Leistungsniveau der Studenten zu verschaffen. Zudem können Dozenten nachhaltig das Lernangebot an die Studenten anpassen. (An et al., 2016)


Abbildung 29: SLHw openLCMSI (Eigene Darstellung)


6.1.3 Bewertung der Kernziele

Interoperabilität

Die Interoperabilität wird durch die Adapter der TPMs gewährleistet. In der aktuellen MI wurden TPMs für die MWAs LMS, LRS und COM integriert.

In der Abbildung 29 [S.78] wird ersichtlich, dass Moodle 2, Moodle 3, Learning Locker und ein SLHw-internes Repository integriert werden.


Die Entwicklung von MWAs sowie eine sinnvolle Kapselung von Funktionalitäten ist ein iterativer Entwicklungsprozess. In weiteren Versionen der MI werden die LCMS-Funktionalitäten der LMS API entkoppelt und in eine eigenständige MWA überführt.

Dynamisierung von Verbindungswegen


Ein Middleware Consumer ist in der Lage, mehrere Routes einer MWA zu besitzen. Soll die Lernbegleiter-App einen Zugriff auf Drittanbieteranwendungen verschiedener Institutionen ermöglichen, ist zu entscheiden, welche Route zu einer Institution gehört.

Datenhoheit des Nutzers


Eine Ausnahme besteht, wenn die Clientanwendung eine andere Institution anspricht und Daten überträgt.

Authentifizierung von Clientanwendungen

Jede Clientanwendung benötigt Zugangsdaten für einen Middleware Consumer, um Dienstleistungen der MI zu nutzen. Jeder Clientanwendung wird daher ein Middleware Consumer zugeordnet.

In der Abbildung 29 [S.78] werden die Editoren von dem Beuth Consumer vertreten. Der Lernbegleiter und die Recommendation-Engine nutzen den Middleware Consumer des Fraunhofer FOKUS.


Der Lernbegleiter sollte auf ein Backend zugreifen, welches im Besitz des Tokens ist und mit der MI kommuniziert. Auf diese Weise kann der Token nicht von Endnutzern ausgelesen und zweckentfremdet werden.

Branchenübergreifende Nutzung

Die MI integriert derzeit keine der E-Learning-Branche fremden MWAs. In dem Architekturdesign werden alle Schnittstellen als Microservices angeboten.

Es besteht kein direkter Zusammenhang zwischen den MWAs. Eine branchenfremde Integration ist somit möglich. Eine Integration von Dienstleistungen der E-Health-Branche wäre eine mögliche Erweiterung.
6.2 Quantitative Evaluation

Bei der quantitativen Evaluation werden die qualitativen Beobachtungen weitestgehend vermieden und Softwarekriterien, wie beispielsweise Bearbeitungszeiten, ermittelt.

Im Rahmen dieser Arbeit erfolgten Performance-Messungen (Monitoring) in einer kontrollierten Testumgebung.

Abbildung 30: Test Environment - Middleware API (Eigene Darstellung)

6.2.1 Die Testumgebung

Die Testumgebung befindet sich in einem Subnetz der Beuth Hochschule für Technik Berlin. Sämtliche Latenzzeiten und Latenzdifferenzen sind zwischen den verschiedenen Servern minimal. Es werden verschiedene Rechner für die Drittanbieteranwendungen, die MI und die Clientanwendung bereitgestellt, um die Performance nicht zu beeinflussen.

Die relevanten Hardwarespezifikationen der Rechner sind identisch und werden in der Tabelle 9 [S.81] aufgelistet.

Tabelle 9: Hardware specifications - Server (Eigene Tabelle)

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>I7-3770 @3.4 GHz (8 virtual cores)</td>
</tr>
<tr>
<td>RAM</td>
<td>16 GB</td>
</tr>
<tr>
<td>Harddrive</td>
<td>SSD 250 GB</td>
</tr>
<tr>
<td>Network Adapter</td>
<td>Intel(R) 82579LM</td>
</tr>
</tbody>
</table>
Für die MI wird der Applicationserver *Wildfly 9* im *standalone-mode* in der Standardkonfiguration bereitgestellt. Es wurden keine Optimierungen vorgenommen, um die Anzahl der Threads (Clients) und Speichernutzung an das Zielsystem anzupassen.


Die Testumgebung erfüllt demzufolge keinen Produktionsstandard und die Auswertungen der folgenden Testszenarien dienen als Richtwert. Die quantitative Evaluation ist als vorläufig (preliminary) anzusehen.

### 6.2.2 Testszenarien zur Ermittlung der Performance von MWAs


Aufgrund der physikalischen Gegebenheit, dass eine *Middleware* einen *Overhead* verursacht, sind längere Antwortzeiten gegenüber einer direkten Nutzung einer *Drittanbieteranwendung* zu erwarten. Dabei sind die Faktoren, die einen direkten Einfluss auf die Performance haben, zu ermitteln.

**Testszenario 1: Ping**

Das Testszenario *Ping* soll ermitteln, wie lange der Applikationsserver *Wildfly* benötigt, um eine REST-Schnittstelle zu verarbeiten. Hierbei wird kein Programmcode der MI verwendet. Im späteren Verlauf erfolgt eine Gegenüberstellung mit der Verarbeitungszeit, wenn der Programmcode der MI miteinbezogen wird.

**Testszenario 2: Third Party Module Lookup**

Das Testszenario *Third Party Module Lookup* ermittelt die Verarbeitungszeit, um eine geeignete Implementierung für eine *Drittanbieteranwendung* aufzurufen. Der Aufruf erfolgt anhand von *Route*-Informationen.

Zu beachten ist, dass die *Route* nicht von dem *Hub* angefordert wird. In diesem Szenario wird das *Route Caching* verwendet und die Kernimplementierung vollständig in die MI miteinbezogen.
6 Evaluation

Testszenario 3: Bypass Middleware

Das Testszenario Bypass Middleware soll die Grundlage für einen Vergleich schaffen, um aufzeigen zu können, welchen Einfluss die Verwendung der MI auf die Verarbeitungszeit hat. Aus diesem Grund wird in diesem Szenario eine Drittanbieteranwendung direkt von einer Clientanwendung angefragt.

Testszenario 4: Middleware and Third Party

Das Testszenario Middleware and Third Party nutzt die MI zum Aufbau einer Verbindung zu einer Drittanbieteranwendung. Es werden alle Komponenten der Testumgebung durchlaufen.

6.2.3 Ergebnisse der Testszenarien

Vor der Präsentation der Messungen sind einige Begrifflichkeiten aufzuführen:

Testplan: Der Testplan beschreibt den kompletten Ablauf für die Testumgebung.

Thread-Group: In einer Threadgroup wird das Verhalten der Clients bestimmt, die Anzahl der Anfragen festgelegt und die Datenerfassung (Monitoring) sichergestellt.

Ramp-Up: Der Begriff Ramp-Up stammt aus der Wirtschaftswissenschaft und Wirtschaftsinformatik und beschreibt die Anlaufphase eines Produkts. Im Kontext der Performance-Messung ist hierbei das Wachstum an Threads (Clients) in einem festgelegten Zeitraum gemeint. Wird kein Ramp-Up definiert, werden direkt zu Beginn alle Threads (Clients) ausgeführt.

Elapsed: Der Begriff Elapsed steht für die Dauer von einer Clientanfrage (request) bis zu einer Serverantwort (response).

Execution: Der Begriff Execution steht für die Zeit in Nanosekunden, die der Programmcode der MI benötigt, um eine Clientanfrage vollständig zu verarbeiten.

Ergebnis 1: Die Middleware API (MWA) Performance

Das erste Ergebnis zeigt die Performance einer MWA, ohne eine Drittanbieteranwendung anzufragen.

Es wurden zwei Thread-Groups mit 100 Threads (Clients) erstellt. Jeder Thread führt 100 Iterationen aus und versendet jeweils eine Clientanfrage pro Iteration. Es wurde ein Ramp-Up von 0 gewählt, sodass 100 Clients gleichzeitig Anfragen stellen. Daraus
folgt, dass insgesamt 10.000 Clientanfragen gestellt wurden. Die beiden Thread-Groups erfüllen jeweils die Testszenarien 1 und 2. Jede Thread-Group wurde für die finale Auswertung 10-mal ausgeführt.

Die Tabelle 10 [S.84] zeigt im Detail, wie die MWA auf die Clientanfragen reagiert.

Tabelle 10: Scenarios 1 and 2 with 10.000 Requests - 100 Threads (Eigene Tabelle)

<table>
<thead>
<tr>
<th>Testscenarios</th>
<th>Elapsed[ms]</th>
<th>Execution[ms]</th>
<th>Time[s]</th>
<th>CPU[%]</th>
<th>RAM[mb]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ping (Test 1)</td>
<td>469,71</td>
<td>0,052</td>
<td>54,72</td>
<td>4</td>
<td>180</td>
</tr>
<tr>
<td>TP Lookup (Test 2)</td>
<td>485,99</td>
<td>0,510</td>
<td>55,94</td>
<td>4</td>
<td>180</td>
</tr>
</tbody>
</table>

Es wird ersichtlich, dass die Verarbeitungszeit der MWA (Execution) im Vergleich zu den Latzenzeiten des Netzwerks zu vernachlässigen ist. Ein TPM *lookup* benötigt konstant 0,51 Millisekunden. Die gesamte Dauer zwischen Request und Response beträgt im Durchschnitt 0,48 Sekunden.

Eine Betrachtung der Zeiten zwischen Testszenario 1 und 2 ist nicht zu empfehlen, da die Abweichung in diesem Bereich aufgrund der überwiegenden Latenzzzeiten durch das Netzwerk keine eindeutige Aussage ermöglicht.

Allgemein ist festzuhalten, dass der TPM *lookup* die gesamte Performance nicht beeinflusst.

Werden die Testszenarien 1 und 2 unter denselben Bedingungen mit 250 Threads (Clients) statt den bisherigen 100 Threads (Clients) ausgeführt, ist in der Tabelle 11 [S.84] kein drastischer Einbruch in der Performance oder eine Hardwareauslastung erkennbar.

Tabelle 11: Scenarios 1 and 2 with 25.000 Requests - 250 Threads (Eigene Tabelle)

<table>
<thead>
<tr>
<th>Testscenarios</th>
<th>Elapsed[ms]</th>
<th>Execution[ms]</th>
<th>Time[s]</th>
<th>CPU[%]</th>
<th>RAM[mb]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ping (Test 1)</td>
<td>666,39</td>
<td>0,052</td>
<td>84,39</td>
<td>5</td>
<td>180</td>
</tr>
<tr>
<td>TP Lookup (Test 2)</td>
<td>682,74</td>
<td>0,510</td>
<td>86,76</td>
<td>9</td>
<td>180</td>
</tr>
</tbody>
</table>

Bei insgesamt 25.000 eingehenden Anfragen beträgt die gesamte Dauer für einen Request bis zur Response im Durchschnitt 0,68 Sekunden. Dieselbe Dauer ist auch bei der normalen Nutzung einer REST-API in dem Testszenario 1 ermittelt worden.

Die MWA beanspruchte im Testszenario 2 maximal 9% der CPU und maximal 180 Megabyte Arbeitsspeicher. Für die *Java Virtual Machine* (JVM) wurden 512 Megabyte Arbeitsspeicher freigegeben.
Die Ermittlung des Datendurchsatzes erübrigt sich, da neben den gängigen HTTP-Protokolldaten keine weiteren Daten übertragen wurden.

**Ergebnis 2: Middleware API versus Third Parties**

Das zweite Ergebnis betrachtet den Einfluss der MWAs LRS und LMS. Es ist zu überprüfen, ob die Nutzung einer MWA gegenüber der direkten Nutzung einer *Drittanbieteranwendung*, wie in diesem Fall Moodle und Learning Locker, zu drastischen Performanceeinbußen führt.

Für die Testszenarien 2 und 3 wurden ebenfalls Thread-Groups erstellt. Jede Thread-Group besitzt 50 Threads (Clients) mit jeweils 5 Iterationen. Der *Ramp-Up* ist auf einen Wert von 0 gesetzt. Diese Werte basieren auf einer stufenweisen Reduzierung der Anfragen, bis sowohl Moodle als auch Learning Locker alle Anfragen vollständig verarbeiten konnten.

Es ist anzumerken, dass die *Drittanbieteranwendungen* auf eine Datenbank zugreifen, was die Performance mehrerer Clientanfragen drastisch beeinflusst. Auch die Verwendung von XAMPP und die nicht produktiv einsetzbaren Konfigurationen reduzieren die Leistung der Systeme.

Die Testszenarien 2 und 3 wurden für jedes Setup (Moodle / Learning Locker) 8-mal ausgeführt.

In der Tabelle 12 [S.85] werden die Daten im Detail aufgelistet.

<table>
<thead>
<tr>
<th>Testscenarios</th>
<th>Elapsed[ms]</th>
<th>Execution[ms]</th>
<th>Time[s]</th>
<th>CPU[%]</th>
<th>RAM[mb]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moodle (Test 3)</td>
<td>768,39</td>
<td>n/a</td>
<td>5,197</td>
<td>100</td>
<td>n/a</td>
</tr>
<tr>
<td>Moodle + MWA (Test 4)</td>
<td>902,56</td>
<td>865,07</td>
<td>5,381</td>
<td>100/11</td>
<td>-/180</td>
</tr>
<tr>
<td>Learning Locker (Test 3)</td>
<td>2174,94</td>
<td>n/a</td>
<td>11,71</td>
<td>100</td>
<td>n/a</td>
</tr>
<tr>
<td>Learning Locker + MWA (Test 4)</td>
<td>2224,88</td>
<td>2207,62</td>
<td>13,28</td>
<td>100/8</td>
<td>-/180</td>
</tr>
</tbody>
</table>

Ohne die Verwendung der MWA (LMS) beträgt die Dauer bei insgesamt 250 eingehenden Clientanfragen bei der Verwendung von Moodle im Durchschnitt 0,76 Sekunden. Wird die MWA für das Routing und die Datentransformation genutzt, steigt die Dauer auf 0,9 Sekunden.


### Tabelle 13: Performance Middleware APIs vs. Third Parties (Eigene Tabelle)

<table>
<thead>
<tr>
<th>Testscenarios</th>
<th>Bytes per Response</th>
<th>Max. Time [s]</th>
<th>Number of Request</th>
<th>Performance [MB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moodle (Test 3)</td>
<td>27987 (Course)</td>
<td>5,19</td>
<td>250</td>
<td>1,34</td>
</tr>
<tr>
<td>Moodle + MWA (Test 4)</td>
<td>27987 (Course)</td>
<td>5,38</td>
<td>250</td>
<td>1,30</td>
</tr>
<tr>
<td>Learning Locker (Test 3)</td>
<td>14615 (Statements)</td>
<td>11,71</td>
<td>250</td>
<td>0,31</td>
</tr>
<tr>
<td>Learning Locker + MWA (Test 4)</td>
<td>14615 (Statements)</td>
<td>13,28</td>
<td>250</td>
<td>0,27</td>
</tr>
</tbody>
</table>

Für den Datendurchsatz wird die maximale Verarbeitungsdauer (Worst Case) aller Clientanfragen aus den 8 Testversuchen für jedes Szenario ermittelt. Bei *Moodle* ergibt sich eine Differenz des Datendurchsatzes von etwa 3%. Die Messungen für *Learning Locker* ergeben eine Differenz von etwa 13%. (siehe Abbildung 31 [S.86])

![Abbildung 31: Performance - Middleware APIs vs. Third Parties (Eigene Darstellung)](image_url)
Mit den erfassten Daten ist schwer zu ermitteln, wie stark die Performance durch die MWA beeinflusst wird. Das Verhältnis zwischen Latenzzeit und der Verarbeitungszeit der MWA wird in der Testumgebung verzerrt, da die Latenzzeiten durch das Subnetz sehr gering sind.

Letztendlich bestätigt diese Erkenntnis, dass in einer realen Umgebung mit höheren Latenzzeiten die Datentransformation und der Third Party Lookup (MWA execution time) sehr wahrscheinlich zu vernachlässigen sind.

Abschließend ist anzumerken, dass auf Basis der vorliegenden Daten die MWA grundsätzlich der Overhead-Problematik von *Middlewares* unterliegt. Basierend auf dieser Tatsache sollte eine sehr gute Netzwerkinfrastruktur angestrebt werden, um den Overhead zu minimieren.


Es folgt in Tabelle 14 [S.87] eine finale Auflistung der Faktoren, die Einfluss auf die Performance nehmen können:

<table>
<thead>
<tr>
<th>No.</th>
<th>Performance factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Third Party-Application performance and API compatibility</td>
</tr>
<tr>
<td>2</td>
<td>Network infrastructure and server setup</td>
</tr>
<tr>
<td>3</td>
<td>Datamodel (DTO / Entity)</td>
</tr>
<tr>
<td>4</td>
<td>Third Party Module (adapter)</td>
</tr>
<tr>
<td>5</td>
<td>Third Party Module Lookup</td>
</tr>
</tbody>
</table>

### 6.3 Zusammenfassung

In der Evaluation wurden qualitative und quantitative Softwarekriterien ausgewertet.


Die Interoperabilität zu verschiedenen Lernsystemen, wie *Moodle 2* und *Moodle 3* (LMS) sowie *Learning Locker* (LRS), wird durch TPMs ermöglicht.
Die Dynamisierung von Verbindungswegen, um die verschiedenen Institutionen der drei Kurse ansprechen zu können, ist durch das sogenannte Routing innerhalb der MI realisiert worden.

Nach Einwilligung des Nutzers werden Daten über die MI aus verschiedenen Institutionen bezogen. Eine Aggregation der Daten erfolgt clientseitig und nicht zwischen den Drittanbieteranwendungen.

Entwickler müssen sich über einen Consumer Token an der MI authentifizieren. Der Zugriff auf Routes und somit Drittanbieteranwendungen wird auf diese Weise kontrolliert.

Der branchenübergreifende Einsatz wird durch die MWAs, die auf Microservices basieren, ermöglicht. Es erscheint vielversprechend, eine MWA für Fitnesstracker (E-Health) neben der E-Learning-Branche zu integrieren.


Es wurde die normale Nutzung einer Drittanbieteranwendung mit der Nutzung der MI verglichen. Ohne die Anbindung einer Drittanbieteranwendung konnten 250 Clients mit jeweils 100 Anfragen ohne auffallende Probleme verarbeitet werden.

Der Overhead der Middleware hinsichtlich der Performance (Datendurchsatz) ergibt in der kontrollierten Testumgebung bei Moodle eine Differenz von etwa 3%. Die Messungen für Learning Locker weisen eine Differenz von etwa 13% auf. Die Abweichung ist trotz der Latenzzeiten durch das Netzwerk angemessen, sodass die eigentliche Performance der MI die nötigen Anforderungen des SLHw-Projekts erfüllt.
7 Zusammenfassung

Das Ziel dieser Arbeit war die Entwicklung einer Middleware Infrastructure (MI), die vorerst für verschiedene Drittanbieteranwendungen der E-Learning-Branche zentrale und einheitliche Schnittstellen bereitstellt.


7.1 Fazit


Während der Recherchen wurde oftmals das Problem geäußert, dass die Standards beispielsweise von IMS oder ADL nicht ausreichend in allen E-Learning-Angeboten reibungslos eingesetzt werden können und somit den heterogenen Markt nicht vollständig abdecken.


Im Fokus der Recherche standen Architekturansätze nach dem SOA-Paradigma, welches im Wesentlichen das Registrieren (register), das Auffinden (find), das Integrieren (bind) und das Ausführen (execute) von Services beschreibt (Zhu, 2005). Des Weiteren wurde zum Zwecke der branchenübergreifenden Nutzung der MI die Idee von Microservices aufgegriffen.


Aufgrund dieser umfangreichen Anwendungsszenarien und der Einbindung verschiedener Institutionen wurde für die MI ein Konzept verfolgt, bei dem dynamische Verbindungswege (Routes) einen Zugriff auf Drittanbieteranwendungen der Institutionen ermöglichen.

Entwickler einer Clientanwendung müssen sich über einen *Consumer Token* an der MI authentifizieren. Der Zugriff auf Routen und somit Drittanbieteranwendungen wird auf diese Weise kontrolliert.

Ein Mehrwert im Vergleich zu den in dieser Arbeit vorgestellten verwandten serviceorientierten Ansätzen ist das Verhindern einer direkten Kommunikation zwischen Institutionen. Daten werden, wenn ein Nutzer einwilligt, über die MI aus verschiedenen Institutionen bezogen und die *Aggregation* der Daten erfolgt clientseitig. Auf diese Weise wird eine Alternative für die Datenhoheit eines Nutzers geboten.


Zum einen wurde die Leistungsfähigkeit der MWA ohne Drittanbieteranwendung getestet. Es wurden 250 Clients mit jeweils 100 Anfragen ohne auffällende Probleme verarbeitet. Dies entsprach 25.000 gleichzeitigen Clientanfragen. Der Mehraufwand durch das Routing betrug bei einer Anfrage etwa 0,5 Millisekunden.


7.2 Ausblick


Die aktuelle MI liefert über einen Hub alle Routen, die für einen Middleware Consumer autorisiert sind. Es fehlt jedoch eine Zuordnung zu der entsprechenden Institution. Hierbei könnte ein Konzept zum Einsatz kommen, bei dem zu jeder Institution eine Sammlung von zugehörigen Routen ausgeliefert wird.


Abbildung 32: Marktanteile Hersteller Wearables 2014/2015 (Statista, 2015)


Seit 2016 ermöglicht das E-Health-Gesetz – Gesetz für sichere digitale Kommunikation und Anwendungen im Gesundheitswesen – die Entwicklung digitaler Infrastruktur-

Im Kontext dieser Arbeit wäre die MI eine Einsatzmöglichkeit für die telemedizinische Kommunikation. Technische Sicherheitsstandards des E-Health-Gesetzes sind bei einer Weiterentwicklung zu ermitteln. Die telemedizinischen Apps und Patienteninformationssysteme wären in diesem Fall die Clientanwendungen und die Ärzte, technisch gesehen Institutionen mit Drittanbieteranwendungen, die einen Zugriff auf Patientendaten ermöglichen. Die Datenhoheit des Nutzers ist auch in der E-Health-Branche ein wichtiges Kriterium.

7.3 Schlusswort

Es wurde der erste Meilenstein erreicht, durch die konzipierte Middleware Infrastructure (MI) Dienstleistungen der E-Learning-Branche zentral zu vereinen.


Der Autor dieser Arbeit sieht in der Nachnutzung ein Potenzial, wenn die Adapter für Drittanbieteranwendungen sowie neue Branchen modular durch eine Community eingepflegt werden. Auf diese Weise könnten auch die weniger genutzten Drittanbieteranwendungen und eine Vielzahl von Branchen Berücksichtigung finden.

Die Verwendung der MI begünstigt in Clientanwendungen einen hohen Grad an Kompatibilität sowie eine hohe Erreichbarkeit von E-Learning-Angeboten, ohne einen erhöhten Entwicklungsaufwand zu verursachen.

Die MI wird voraussichtlich Ende September 2017 an der Handwerkskammer Berlin für eine Nachnutzung aufgesetzt und quelloffen zur Verfügung gestellt.
Literaturverzeichnis


IETF. The 'Basic' HTTP Authentication Scheme This. *Internet Engineering Task Force (IETF)*, pp. 1–15, 2015.


Abbildung 33: LRS Statement (Eigene Darstellung)
Abbildung 34: LMS Course (Eigene Darstellung)
C Hub Management

Abbildung 35: Edit Middleware Consumer (Eigene Darstellung)
Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Unterschrift:  Ort, Datum: