BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN

University of Applied Sciences

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

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

ACP	Adaptive Contextual Portal
ADL	Advanced Distributed Learning
API	Application Programming Interface
CC	Consumer Credentials
CDI	Context and Dependency Injection
COM	Communication
CRM	Customer-Relationship-Management
CRUD	Create-, Read-, Update- und Delete-Methoden
DAO	Data Access Object
DTO	Data Transfer Object
EJB	Enterprise JavaBeans
ELF	The E-Learning Framework
ESB	Enterprise Service Bus
FUMES	Federated User Modeling Exchange Service
GGF	Global Grid Forum
HR	Human Resource
HTTP	Hypertext Transfer Protocol
IAF	IMS Abstract Framework
IDA	Independently Deployed Application
IMS CC	IMS Common Cartridge
IMS LTI	IMS Learning Tools Interoperability
IMS QTI	IMS Question & Test Interoperability
IoC	Inversion of Control
IRI	Internationalized Resource Identifier
JNDI	Java Naming and Directory Interface
JSON	JavaScript Object Notation
LA	Learning Analytics
LCMS	Learning Content Management System
LDAP	Lightweight Directory Access Protocol
LMS	Learning Management System
LRS	Learning Record Store
LTI	Learning Tool Interoperability
MI	Middleware Infrastructure
MIP	Middleware Infrastructure Provider
MOOC	Massive Open Online Course
MWA	Middleware API
OCCS	Open Corpus Content Service
OGSA	Open Grid Service Architecture
OSS	Open-Source-Software
PLE	Personal Learning Environment
REST	Representational State Transfer
RLO	Reusable Learning Object
SCORM	Sharable Content Object Reference Model
SOA	Serviceorientierte Architektur
TKB	Technical Knowledge Database
TPM	Third Party Module
UC	User Credentials
URL	Uniform Resource Locator
USR	User
VF	Virtual File
VFS	Virtual File System
VO	Virtuelle Organisation
WAR	Web Application Archive
WS	Web-Service
XML	Extensible Markup Language

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 Java-EE-Entwurfsmusters *Business Delegate* ein geeignetes *Data Access Object*.

Data Access Object Entwurfsmuster, das über ein Objekt den Zugriff auf eine Datenerhaltungsschicht 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.

Hadoop Framework zum Entwickeln von Software, die ein Cluster-Dateisystem verwendet und große Datenbestände nach dem MapReduce-Programmiermodell verarbeitet.

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 zwischenspeichert.

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

IMS Common Cartridge Spezifikation des IMS Global Learning Consortiums und beschreibt ein Format zum Erstellen sowie Teilen von digitalen Bildungsinhalten.

IMS Question and Test Interoperability Spezifikation des IMS Global Learning Consortiums und beschreibt ein Format zum Erstellen sowie Teilen von Online-Fragen, Tests und Quizzes.

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 Softwarearchitekur zur Erstellung mehrschichtiger Unternehmenssoftware, wie beispielweise im Rahmen dieser Arbeit die *Middleware Infrastructure*.

JavaScript Object Notation ein menschen- 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 Javscript-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.

Learning Tool Interoperability Spezifikation des IMS Global Learning Consortiums und beschreibt einen Standard, um *Rich Client Applications* als Lernanwendungen auszuliefern (LTI Provider) und diese in separat bereitgestellten Anwendungen zu integrieren (LTI Consumer).

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 Datenerhaltungsschicht.

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

Representational State Transfer vereinfacht ein Architekturmodell, das über die HTTP-Methoden GET, POST, PUT und DELETE das Lesen, Erstellen, Aktualisieren und Löschen von Ressourcen zustandslos über *Webservices* ermöglicht.

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

Rich Client Application textitClientanwendung, die Anwendungslogik 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.

Single Sign-On Zugriff auf mehrere Systeme oder Anwendungen durch einmalige Authentifizierung eines Nutzers.

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 aussagt, 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*.

Virtuelle Organisation Eine Menge von Individuen oder Institutionen bilden nach den Richtlinien der gemeinsamen Nutzung von Ressourcen eine Virtuelle Organisation.

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

Webservice ermöglicht eine *Machine-to-Machine*-Kommunikation über HTTP und HTTPS in verteilten Systemen.

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

Der Autor dieser Arbeit beschäftigte sich während des Studiums intensiv mit *Backend*-Architekturen. Durch die Beteiligung an den Forschungsprojekten *fMOOC – Interaktion von Senioren mit tragbaren Fitnesstrackern in integrierter MOOC-Plattform –* (Buchem et al., 2015) und *Smart Learning – Medieneinsatz in der handwerklichen Weiterbildung –* (Krauss et al., 2016b) wuchs ein besonderes Interesse an *Middleware*-Architekturen. Eine Motivation, die sich im Laufe der Projekte entwickelte, war die Konzeption einer *Middleware*, welche in erster Linie wie ein gewöhnliches *Representational State Transfer Application Programming Interface* (REST-API) agiert.

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.

In dem SLHw-Projekt trat ein ähnliches Szenario auf. Statt Vitaldaten wurden Lerninhalte behandelt. Es gibt viele Bildungseinrichtungen, die eigene Online-Kurse bereitstellen. Jede Bildungseinrichtung verwendet eine eigene Lernplattform und somit andere Schnittstellen und Lerninhalte. Die Art der Daten und die Informationen, die vermittelt werden sollen, sind auch in diesem Fall identisch. Letztendlich wird in dieser Arbeit das Ziel verfolgt, für verschiedene Branchen jeweils einheitliche Schnittstellen für mehrere Anbieter bereitzustellen. Die Art der Implementierung soll dieselbe Komplexität sowie dieselben Merkmale aufweisen, wie die Integration für nur einen Anbieter. Ein Fokus ist im Rahmen dieser Arbeit die E-Learning-Branche sowie die Nutzung der zentralen Schnittstellen durch eine Lernbegleiter-App.

[1] Duden Stand Februar 2017

1.2 Problemstellung

Werden speziell E-Learning-Angebote betrachtet, so besteht zwischen den verschiedenen Anbietern oftmals ein ähnlicher Funktionsumfang. Aus Sicht der Entwickler ergibt sich jedoch das Problem, Standards zu finden, die von allen Anwendungen berücksichtigt werden. Auch bei einer neuen Softwareversion können die Schnittstellen unterschiedliche Implementierungen aufweisen.

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.

Bei der Verwendung von bestehenden serviceorientierten *Middleware*-Architekturen, die eine *Interoperabilität* ermöglichen, wird oft eine Kommunikation zwischen verschiedenen Dienstleistern angestrebt. Aus Sicht der Datenhoheit würden in der E-Learning-Branche Daten eines Nutzers potenziell an weitere Bildungseinrichtungen übermittelt werden. Ein sekundäres Problem ist somit, dass Daten nicht nur für den Nutzer aufbereitet werden. Der Datenschutz ist auch in der E-Learning-Branche ein Thema, das nicht zu vernachlässigen ist.

1.3 Zielsetzung

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

Interoperabilität: Anstatt die Kompatibilität von Schnittstellen durch Standards auf Ebene der Anwendungen zu gewährleisten, sollen zentrale *Middleware*-Schnittstellen die *Interoperabilität* erfüllen. Die dazu benötigte Datentransformation wird durch Adapter realisiert, die in die MI integriert werden. Es soll vermieden werden, dass beispielsweise bei der Integration eines weiteren *Learning Management System* (LMS) erneut ein hoher Implementierungsaufwand in der *Clientanwendung* entsteht.

Dynamisierung von Verbindungswegen: Neben der Schnittstellen-*Interoperabilität* steht auch die Dynamisierung von Verbindungswegen zu verschiedenen Instanzen einer Anwendung im Vordergrund. In der Problemstellung wurde ein lebenslanges Lernen angeführt. In diesem Kontext handelt es sich um Anwendungen von mehreren Bildungseinrichtungen, die von einem Lerner beansprucht werden.

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.

Authentifizierung von Clientanwendungen: Die Nutzung von Verbindungswegen soll über eine zentrale Verwaltungskomponente geregelt werden. Hierbei werden die *Clientanwendungen* authentifiziert. Daraus folgt, dass eine *Clientanwendung* nicht alle verfügbaren Anwendungen der Bildungseinrichtungen über die MI nutzen kann.

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

Zu Beginn dieses Kapitels wird die Bedeutung von *Learning Management Systems* (LMS) in der E-Learning-Branche erläutert. Anschließend erfolgt eine allgemeine Marktübersicht verschiedener LMS, um die Heterogenität dieser zu verdeutlichen. Eine historische Betrachtung der LMS zeigt den Wandel beziehungsweise den Bedarf an flexiblen interoperablen serviceorientierten Ansätzen aufgrund der vorherrschenden monolithischen Systeme in der heterogenen Systemlandschaft. Des Weiteren wird das Potenzial von serviceorientierten Ansätzen in der E-Learning-Branche verdeutlicht. Abschließend werden serviceorientierte Ansätze verwandter Arbeiten beschrieben.

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

Die E-Learning-Branche wird besonders durch LMS bestimmt. Die Aufgabe von LMS ist es, eine Vielzahl von Funktionalitäten anzubieten, welche ein digitales Lernen ermöglichen. Die Anforderungen an ein LMS können sehr verschieden sein. Aus diesem Grund haben sich komplexe monolithische Systeme gebildet. Im klassischen Sinne werden LMS für das Nutzer- und Kursmanagement verwendet.

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.

Hinsichtlich der Lerninhalte wurden neben den Grundfunktionalitäten eines LMS auch Autorenwerkzeuge entwickelt. Sowohl das Erstellen und die Bearbeitung als auch die Wiederverwendbarkeit von Lerninhalten (*reusable learning objects*, RLO, Baumgartner (2004)) stehen hierbei im Vordergrund. Neben den RLO können Kursinhalte wie Powerpoint-Präsentationen, Texte, Videos und Audiodateien über Kurse veröffentlicht werden. Systeme mit diesem Funktionsumfang werden als *Learning Content Management System* (LCMS) bezeichnet.

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.

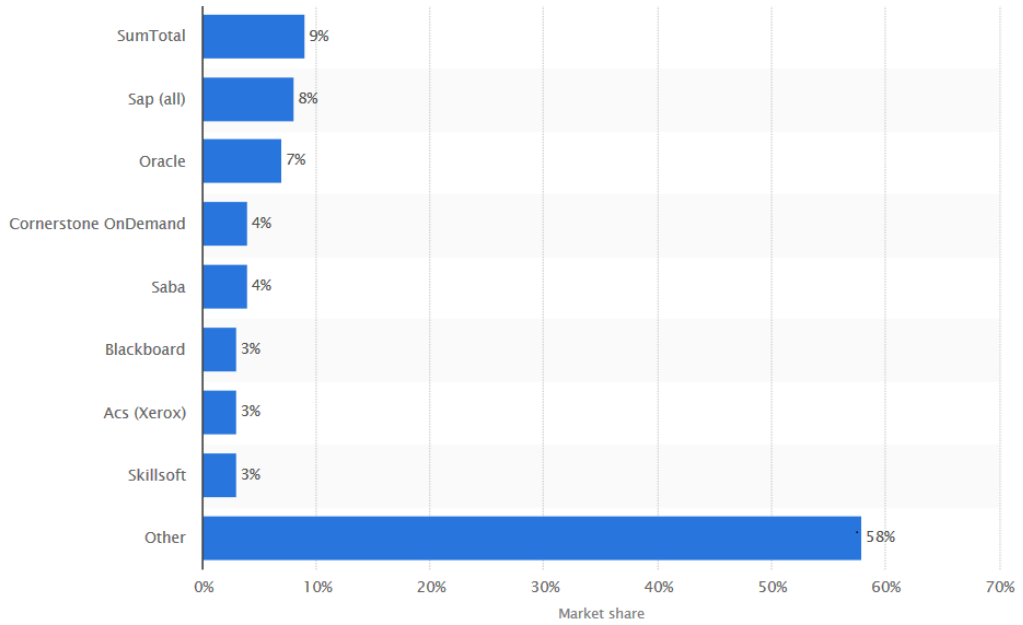

Abbildung 1: Share of LMS market worldwide in 2013, by vendor (Statista, 2013)

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), *Ilias*, *Eduplone*, *SAKAI* und *WebCT* (Zedan and Al-Ajlan, 2005).

Durch das hohe Aufkommen von LMS entsteht eine starke Heterogenität. Der Bedarf an *Interoperabilität* und einheitlichen Standards wächst stetig. Es gibt verschiedene Organisationen, Initiativen und Konsortien, die Empfehlungen und Standards für die E-Learning-Branche entwickeln. Es folgt in Tabelle 1 [S.18] eine Auflistung der bekanntesten Vertreter.

Tabelle 1: E-Learning Standards (Eigene Tabelle)

IMS Global Learning Consortium (IMS)	IMS Metadata, IMS Content Packaging, IMS Question & Test Interoperability
Advanced Distributed Learning Network (ADLNet)	Sharable Content Object Reference Model (SCORM)

Standards haben den Vorteil, dass wesentliche Aspekte einer *Domäne* konkretisiert und spezifische Funktionalitäten abstrahiert werden. Auf diese Weise sind die aufgelisteten Standards richtungsweisend. Des Weiteren wird die Akzeptanz von LMS verbessert, wenn bekannte Standards implementiert werden. Ein wesentlicher Nachteil besteht darin, dass es keine Verbindlichkeiten gibt, wie systematisch ein Standard umgesetzt wird. Die Verantwortung für die Kompatibilität tragen oft die Anbieter eines LMS.

> In Clarke (2005) wurde die Content-Interoperabilität einer Vielzahl von Lernplattformen in Bezug auf den SCORM-Standard untersucht. Es zeigte sich, dass derzeit der Import und Export von Lernobjekten, die den SCORM-Standard erfüllen, zwischen den Lernplattformen nicht immer reibungslos funktioniert. (Frankfurth and Schellhase, 2006)

2.3 Eine historische Betrachtung von LMS

In Dagger et al. (2007) wird der historische Verlauf von LMS in drei Generationen unterteilt. Die erste Generation (etwa 1993) von LMS fokussierte sich auf die Bereitstellung und die Interoperabilität von Inhalten, die für einen bestimmten Zweck entworfen wurden, wie beispielsweise ein bestimmtes Kursformat. Die zweite Generation (etwa 1999) basierte weiterhin auf den vorhergehenden Konzepten. Eine Verbesserung war, dass Inhalte nicht nur bereitgestellt, sondern auch die Wiederverwendbarkeit von Lernobjekten und das Teilen von Lernprofilen ermöglicht wurden. Eine weitere und besonders auch für diese Arbeit wichtige Optimierung war die Modularisierung der LMS. Durch das modulare Design war es einfacher, neue Funktionalitäten zu integrieren.

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)

Die dritte Generation (engl. next generation) von LMS bestätigt den langfristigen Wandel von monolithischen Systemen zu *serviceorientierten Architekturen* (SOA). Im Rahmen dieser Arbeit wird ausschließlich *Moodle* als *Drittanbieteranwendung* integriert. Für die Konzeption der MI sollten jedoch auch aktuelle Trends in der E-Learning-Branche betrachtet werden. Die spätere MI sollte die Paradigmen der aktuellen Trends berücksichtigen, um die eigenen Schnittstellen zukunftsorientiert gestalten zu können.

Bei dem ersten Trend handelt es sich um das *Personal Learning Environment* (PLE). In diesem Forschungsfeld steht der Lerner im Zentrum des Systems. Van Harmelen (2006) beschreibt einige Aspekte, die den Begriff PLE prägen. Im Vordergrund stehen das lebenslange Lernen und der daraus resultierende Bedarf an standardisierten Schnittstellen zu verschiedenen Lernsystemen. Diese setzen voraus, dass die Lernprofile eines Nutzers in verschiedenen Institutionen abgerufen werden können. Ein Lernender soll die Kontrolle über das E-Learning-System erlangen, sodass ein selbstbestimmtes

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.

Das Ziel ist es, durch Bildung von temporären Gemeinschaften Inhalte und Informationen zu verteilen, um sich gegenseitig mit Wissen und Ideen zu unterstützen (Peters et al., 2015). An dieser Stelle folgt ein Zitat von George Siemens, der als Begründer des Konnektivismus gilt.

> 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).

Im Jahr 2015 wurde der Markt hinsichtlich der MOOC durch die Anbieter *Coursera*, *Open edX*, *Udacity* und *FutureLearn* bestimmt. Die Nutzung der benannten MOOC-Anbieter wird in der Tabelle 2 [S.21] verdeutlicht.

Tabelle 2: Key Figures of Selected MOOC Platforms (Peters et al., 2015)

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

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.

Frankfurth and Schellhase (2006) bewerten das Potenzial von SOAs in der E-Learning-Branche anhand von technologischen, organisatorischen, qualitativen und ökonomischen Kriterien.

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)

Qualitativ würde durch den Einsatz von SOAs eine Orchestration (Kopplung) von Services möglich sein. Verschiedene Bereiche einer Bildungseinrichtung, wie zum Beispiel Bibliothekssysteme, Prüfungsverwaltungssysteme und E-Learning-Systeme, könnten über eine Anwendung zentralisiert werden. (vgl. Frankfurth and Schellhase, 2006)

Ökonomisch sollte es vermieden werden, Projektmittel für Eigenentwicklungen monolithischer Systeme aufzubringen. Vielmehr wäre es wünschenswert, dass E-Learning-Anbieter und Forscher aktiv die Entwicklung spezieller E-Learning-Services begleiten. (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

finden kann. Fowler and Lewis (2015) haben den Versuch unternommen, eine Definition und die Eigenschaften von *Microservices* zu beschreiben. Bisher existiert keine allgemeingültige Definition. Grundlegend ist festzuhalten, dass es sich um voneinander unabhängig auslieferbare Services handelt. Die Abbildung 2 [S.22] stellt einen direkten Vergleich zu einem monolithischen System her.

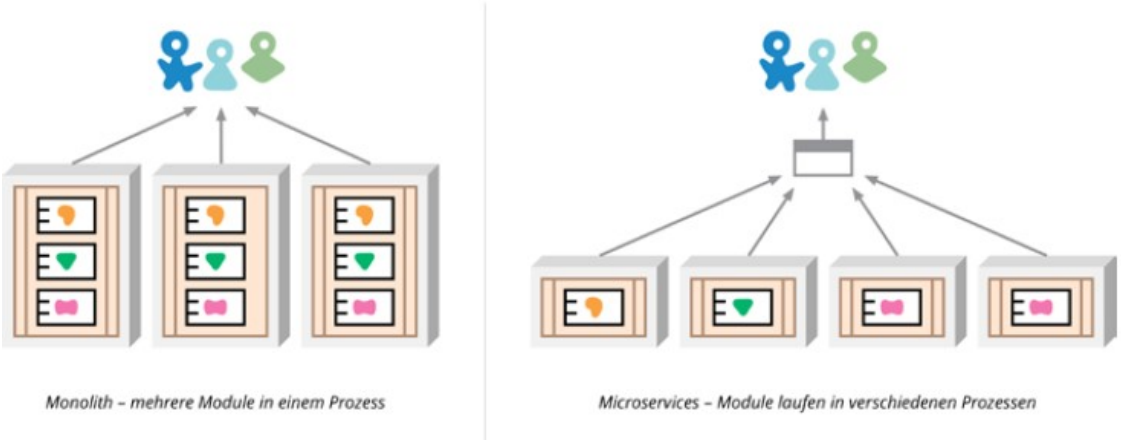

Abbildung 2: Modular deployment (Fowler and Lewis, 2015)

Obwohl beide Ansätze modular sind, unterscheiden sich diese stark voneinander. Ein monolithisches LCMS, wie zum Beispiel *Moodle*, hat eine modulare Schichtenarchitektur und auch das Hinzufügen von Plugins stellt kein Problem dar. Dennoch findet die Kommunikation zwischen den Modulen innerhalb der Applikation statt.

Auch das Bereitstellen von *Webservices* für einen externen Zugriff löst dieses Problem nur bedingt. Die *Webservices* ermöglichen eine *machine-to-machine*-Kommunikation. Es wird aber nicht berücksichtigt, unnötige Abhängigkeiten zu vermeiden.

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.

Abhängig von der Implementierung und dem Anwendungsszenario können *Deployments* auch zur Laufzeit erfolgen. Eine Skalierung durch das mehrfache Ausliefern eines *Microservices* ist ebenfalls gegeben. Anwendungen, die eine *Microservice*-Architektur verwenden, haben gegenüber einem *Monolithen* durch das verteilte System eine höhere Komplexität hinsichtlich der Ausfallsicherheit. Bei einem Ausfall von Services muss sichergestellt werden, dass der *Service Consumer* nur bei den betroffenen Funktionalitäten ausfällt, jedoch nicht das Gesamtsystem.

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.

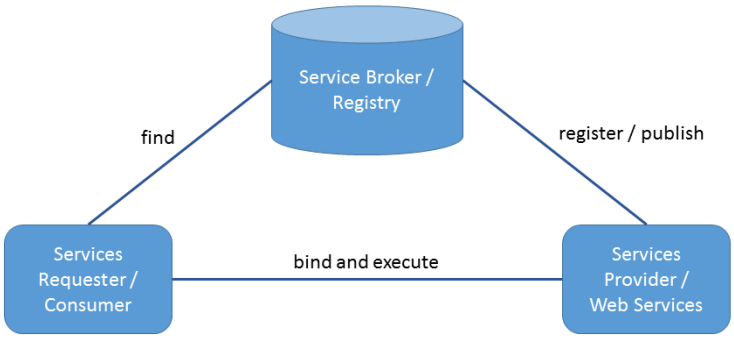

Abbildung 3: find-bind-execute paradigm based on (Zhu, 2005)

Der *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.

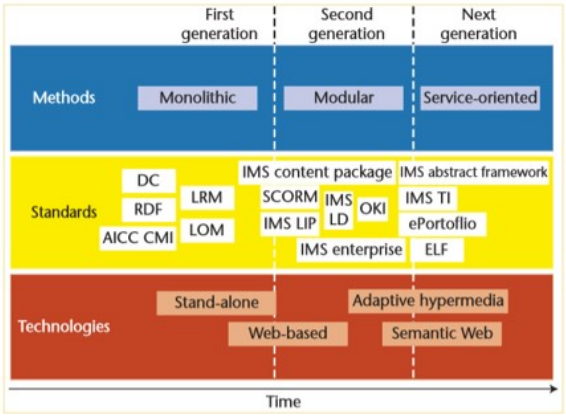

Abbildung 4: Generations of learning management systems (Dagger et al., 2007)

In der Abbildung 4 [S.24] wird erneut der Wandel hinsichtlich der LMS von monolithischen über modulare Systeme zu SOAs verdeutlicht. Es ist unter anderem das *IMS Abstract Framework* (IAF) als SOA Framework abgebildet. Hierbei handelt es sich nicht um ein lauffähiges System. Das IAF ist eine Ansammlung von verschiedenen IMS-Standards und eine Empfehlung, wie eine SOA auf die E-Learning-Branche anzuwenden ist. Das Framework wird iterativ konzipiert und ändert sich entsprechend neuer Anforderungen. Zudem wird das *E-Learning Framework* (ELF) als Implementierung des IAF dargestellt. Weitere Nennungen in der Abbildung 4 [S.24] sind im Rahmen dieses Kapitels nicht relevant.

Grundlegend besteht das IAF aus den Schichten *Application Layer*, *Application Services Layer*, *Common Services Layer* und *Infrastructure Layer*, wie die Abbildung 5 [S.24] verdeutlicht.

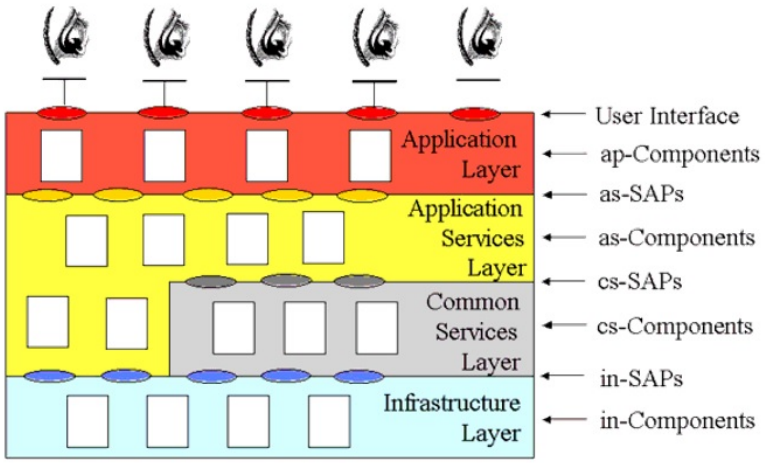

Abbildung 5: The layered model (IMS Global Learning Consortium, 2003)

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

Das *Application Services Layer* stellt E-Learning-typische Services bereit; einige von diesen werden in der Tabelle 3 [S.25] aufgelistet.

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.

Das *Infrastructure Layer* ermöglicht den clientseitigen Zugriff auf entfernte Service Provider über eine *run-time*-Schnittstelle. Jeder Service besitzt zudem einen *Service Access Point* (SAP). Ein Zugriff auf einen Service ist ausschließlich über den SAP möglich, der als *Application Programming Interface* (API) implementiert werden kann.

Tabelle 3: IAF Application Services (IMS Global Learning Consortium, 2003)

Service	Description
ClassAdministration	the management of classes
ContentManagement	management of learning content
LearnerProgressionManagement	management of learning experiences
MetaDataManagement	management of meta-data resources
DigitalRepositoryManagement	management of digital repositories
ProfileManagement	management of a learner's profile, life-long learning log, etc.
EnterpriseServices	management. of course enrolments and learning activities

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.

Es folgten neue Konzepte in der E-Learning-Branche, die sich auf das *Grid-Computing* fokussierten. Eine bekannte grid-basierte SOA ist die *Open Grid Service Architecture* (OGSA), die von dem *Global Grid Forum* (GGF) entwickelt wurde. Das Paradigma

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)

Wird das *Grid Computing* auf die E-Learning-Branche übertragen, könnte es sich bei den geteilten Ressourcen um Repositories mit Lerninhalten handeln. Institutionen könnten beispielsweise Bildungseinrichtungen sein.

Ein Zusammenschluss von Bildungseinrichtungen ermöglicht Lernenden ein großes Angebot an Lerninhalten. Gefördert werden zudem die bereits erwähnten organisatorischen Potenziale von serviceorientierten Ansätzen. Bei einer Lehrkooperation kann durch die Wiederverwendbarkeit von geteilten Lerninhalten die Mehrarbeit reduziert werden.

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.

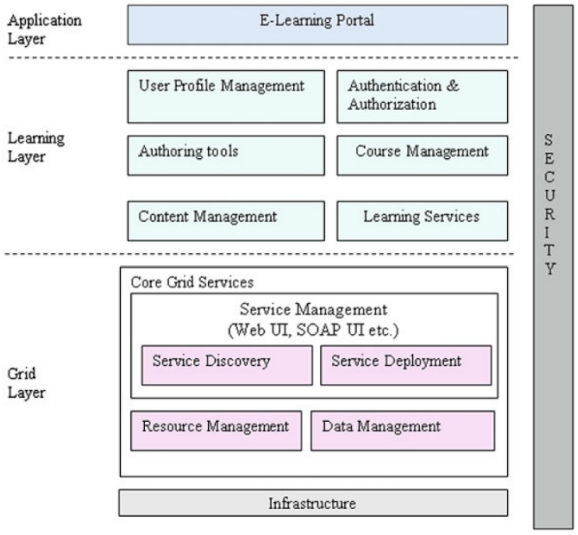

Abbildung 6: Layered architecture of e-Learning Grid (Ivanović and Jain, 2014)

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

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

Das *Grid Layer* (*Grid Middleware*) verfügt über das *Service Management*. Dieses ermöglicht über die Komponenten *Service Deployment* und *Service Discovery* das Registrieren und Auffinden von Services. Das *Resource Management* liefert den Status sowie den Typen von allen *Grid Ressourcen* und das *Data Management* ist für die Distribution der Daten aus einem LMS (*Service Provider*) verantwortlich.

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

Die SOA von Liu et al. (2003) in Abbildung 7 [S.28] zeigt, wie anhand des SOA-Paradigmas mehrere LMS, LCMS und Autorensysteme über *Webservices* kommunizieren können. Jedes System kann potenziell sowohl *Service Consumer* als auch *Service Provider* sein.

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-

teln Servicebeschreibungen zu LCMS-*Service Providern*. Die *Learner Service Discovery Agencies* liefern hingegen Beschreibungen zu LMS-*Service Providern*.

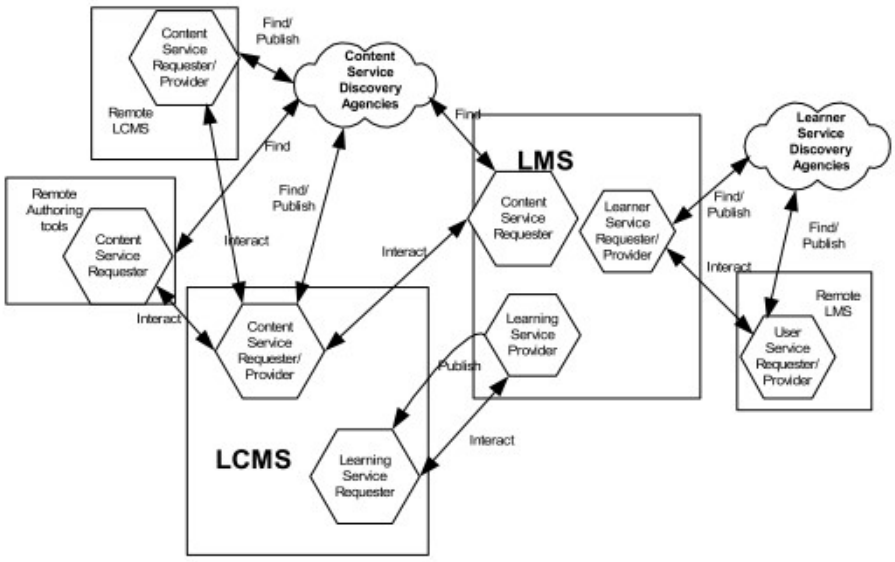

Abbildung 7: Service architecture of e-learning system (Liu et al., 2003)

Ein erweiterter SOA-Ansatz von Aguirre et al. (2008) in Abbildung 8 [S.28] ist es, neben den aus IAF definierten *Application Services* und *Common Services* zusätzlich sogenannte *Joint Degree Services* zu integrieren.

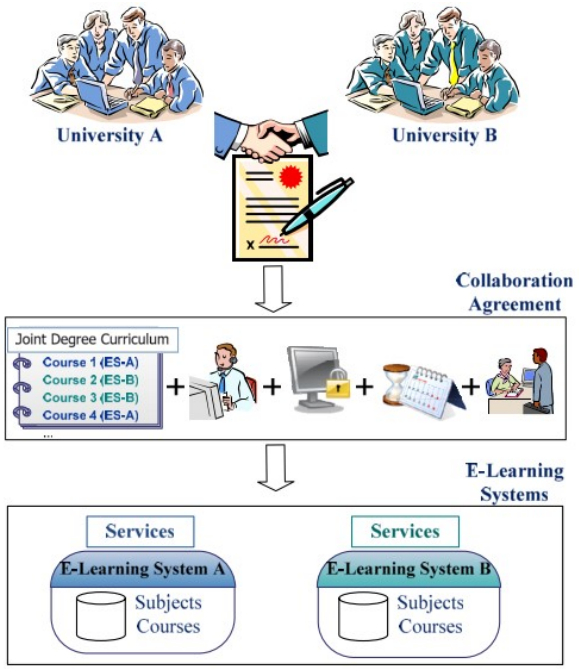

Abbildung 8: Joint Degrees in E-Learning Systems (Aguirre et al., 2008)

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.

An dieser Stelle wird aufgezeigt, dass das IAF als Referenz viel Spielraum hinsichtlich der Kommunikation in einer ohnehin sehr dynamischen SOA offenhält. Die Regelung der Kommunikation wird auch in dem SOA-Ansatz dieser Arbeit konkretisiert werden.

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.

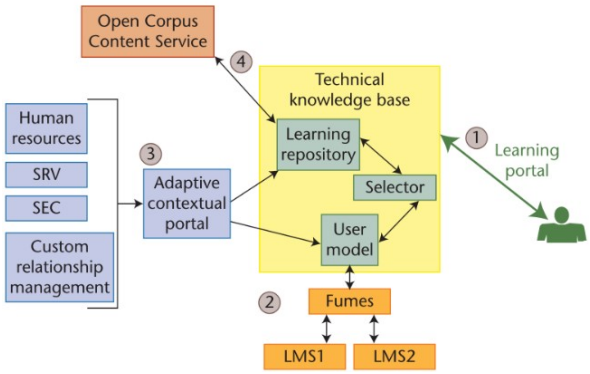

Abbildung 9: Dynamic personalized E-Learning scenario (Dagger et al., 2007)

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.

Ein Ingenieur würde zum Beispiel technische Inhalte bevorzugen. Durch Informationen des CRM erfolgt eine Auswahl der für den Kundenauftrag relevanten Lerninhalte.

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*.

Das nächste Konzept ist eine MOOC-Architektur des non-profit-Anbieters *Open edX* (EdX, 2016). Die in der Abbildung 10 [S.30] dargestellte Architektur besteht aus vier Hauptkomponenten, den *Frontend Tools* und Clients, der *EdX Platform*, den *Independently Deployed Applications* (IDA) und den *Persistence Systems*.

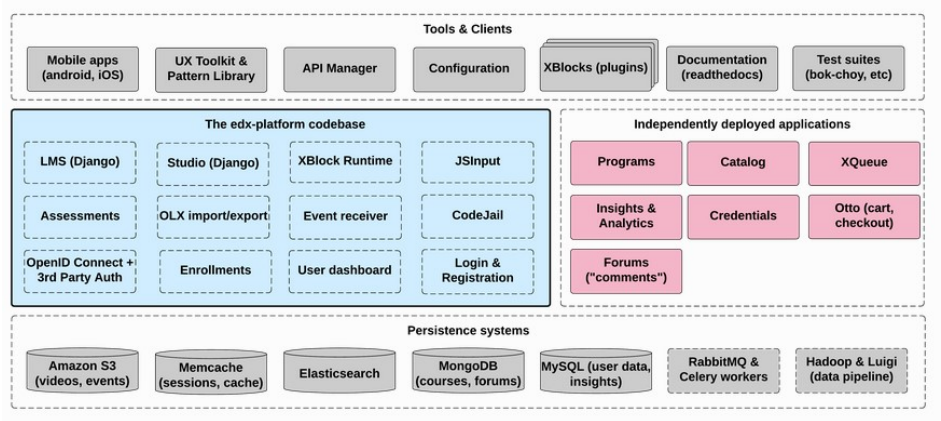

Abbildung 10: Open edX MOOC Architecture (EdX, 2016)

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

Die Kursstruktur wird über eine *MongoDB* persistiert. Jeder Kurs besteht aus *xBlocks*. Durch das Entwickeln von *xBlock-Plugins* ist es möglich, eigene Kurselemente zu integrieren. Alternativ können Inhalte mit *JS-Input* oder dem IMS-Standard *Learning Tool Interoperability* (LTI) eingepflegt werden. Die Bereitstellung von Videos erfolgt über YouTube oder Amazon *S3*.

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* (*Map&Reduce*) 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

Ein *Learning Management System* (LMS) ist für das Nutzer- und Kursmanagement verantwortlich. Wird zusätzlich das Erstellen und Editieren von Lerninhalten ermöglicht, handelt es sich um *Learning Content Management Systeme* (LCMS).

Die Anforderungen an LMS und LCMS können sehr verschieden sein. Im Jahr 2005 gab es mehr als 250 Anbieter für kommerzielles E-Learning und 40 *Open Source Software* (OSS) Angebote. Aufgrund der starken Heterogenität im E-Learning ist ein Wandel von monolithischen Systemen zur SOA ersichtlich.

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

In dem Kapitel *Related Work* wurden verschiedene Architektur-Konzepte der E-Learning-Branche aufgezeigt, von den monolithischen Architekturen bis zu flexiblen SOAs. Sowohl monolithische als auch SOA-basierte Lernsysteme werden aktiv und erfolgreich in Bildungseinrichtungen eingesetzt.

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

Ein Problem, das während der Entwicklung einer *Lernbegleiter-App* auftrat, war die Inkompatibilität zu verschiedenen E-Learning-Angeboten. Im praktischen Umfeld zeigt sich, dass viele Systeme etablierte Standards der E-Learning-Branche nicht zuverlässig berücksichtigen, ein Problem, welches nicht nur die E-Learning-Branche betrifft. Die Gründe für Inkompatibilitäten können sehr verschieden sein. Unterschiede in dem Funktionsumfang, die Unwissenheit über Standards, aber auch die Unabhängigkeit in Hinsicht auf kommerzielle Interessen könnten mögliche Ursachen sein.

Bei der Entwicklung von Lernanwendungen ist für ein interoperables Szenario in mehreren Bildungseinrichtungen ein hoher Aufwand in die Adaption zu investieren. Eine unwahrscheinliche Lösung, die ein Adaptieren verhindert, wäre es, in allen Bildungseinrichtungen das gleiche Lernsystem zu verwenden. Es gibt verschiedene Probleme bei diesem Ansatz, da jedes System andere Anforderungen hat. Möglicherweise distanziert sich eine Institution von Systemen mit *Cloudlösungen*, um datenschutzrechtliche Aspekte zu vermeiden oder es gibt nicht ausreichend finanzielle Mittel für kommerzielle Lösungen.

Das in dieser Arbeit zu konzipierende System soll das Problem der Inkompatibilitäten durch Interoperabilität und domänenspezifische APIs lösen. Hierbei wird eine MI konzipiert, die nicht nur für die E-Learning-Branche spezialisiert ist, sondern branchenübergreifend eingesetzt werden kann. Die Anbindung einer *Drittanbieteranwendung* soll in Form eines Plugins ermöglicht werden. Ein solches Plugin könnte in der E-Learning-Branche eine Implementierung für ein LMS sein.

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.

Die Abbildung 11 [S.33] zeigt eine abstrakt angedeutete MI. Zu den Institutionen gehören die beiden Universitäten (engl. *University*), eine Bibliothek (engl. *Library*) und eine Arbeitsstelle (engl. *Office*). Jede Institution verfügt über mehrere *Drittanbieteranwendungen* für verschiedene Dienstleistungen. Für die Kurs- und Nutzerverwaltung werden die LMS *Blackboard* und *Moodle* eingesetzt. Zur Kommunikation werden die internen Foren von *Moodle* (*MoodleForums*) und *Blackboard* (*BBForums*) aufgeführt. Das Aufzeichnen von Lernaktivitäten erfolgt durch die Analytics-Anwendungen *Piwik* und *Learning Locker*. Bekannt aus dem Kapitel *Related Work* werden über die Arbeitsstelle Mitarbeiter- (HR) und Kundendatenbanken (CRM) bereitgestellt. Die Repositories liefern Lerninhalte einer Institution aus. Als *Clientanwendung* ist eine *Lernbegleiter-App* (engl. Learning Companion App) vorgesehen. Im weiteren Verlauf dieses Kapitels werden die *Middleware*-Schnittstellen, welche für *Learning Record Stores* (LRS), *User* (USR), *Learning Content Management Systems* (LCMS) und *Communication* (COM) verantwortlich sind, erläutert.

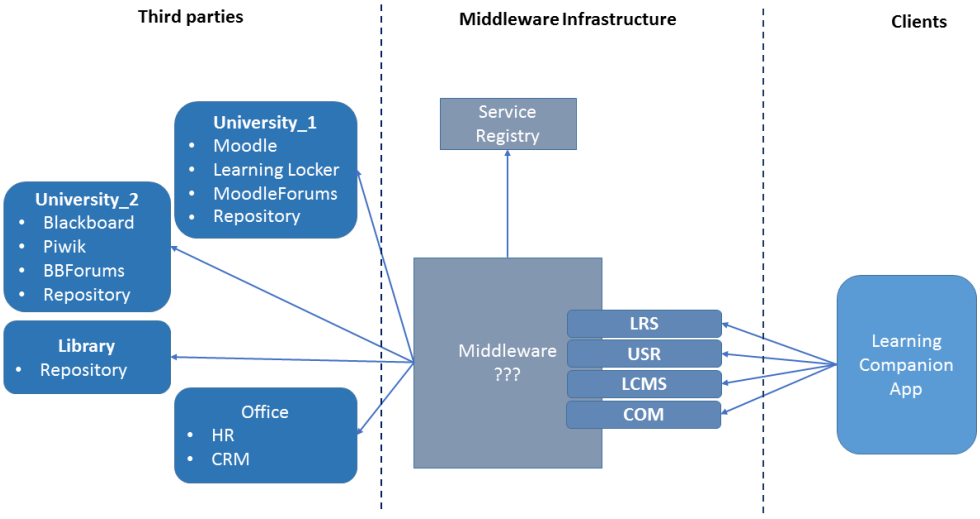

Abbildung 11: Abstract Middleware Infrastructure – MI (Eigene Darstellung)

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.

Van Harmelen (2006) erwähnte im Zusammenhang mit PLE den Bedarf an standardisierten Schnittstellen zwischen verschiedenen Lernsystemen, um ein lebenslanges Lernen zu ermöglichen. Aguirre et al. (2008) betrachten die Integration eines vertraglich gemeinsamen Lehrplans zwischen zwei Universitäten (*Joint Degree*). Dagger et al. (2007) beschreiben ein Szenario, in dem ein Mitarbeiter einer Firma durch die Integration firmenspezifischer Datenbanken im Bereich HR und CRM eine gefilterte Menge an Lerninhalten erhält.

Unter Berücksichtigung des lebenslangen Lernens wäre ein Szenario denkbar, bei dem ein Nutzer an mehreren Institutionen registriert ist. Dieser hat beispielsweise an zwei Universitäten studiert und arbeitet anschließend als Entwickler in einem Büro. Für das Studium und zur Fortbildung wird ausschließlich die Lernbegleiter-App verwendet.

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*.

In dem genannten Beispiel auf Abbildung 11 [S.33] befindet sich eine der beiden Universitäten im Ausland und der Nutzer hat einen Abschluss durch ein *Joint Degree* erlangt. Die LCMS API liefert sowohl die von der Arbeitsstelle bereitgestellten als auch die durch das Studium erhaltenen spezifischen Lerninhalte von Repositories. Neben den spezifischen Lerninhalten aus den Universitäten und der Arbeitsstelle wird über eine Bibliothek Zusatzmaterial zur Verfügung gestellt.

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*.

Die USR API liefert alle Profil-Informationen des Nutzers. Beispielsweise könnten die Personendaten, der Lerntyp, die erlangten Abschlüsse, abgeschlossene Kurse, Handicaps und die aktuelle Berufstätigkeit ermittelt werden. Zur Erfassung der Berufstätigkeit und Aufgabenfelder der Kunden werden die HR- und CRM-Datenbanken der aktuellen Arbeitsstelle abgerufen. Hierbei ist anzumerken, dass die Nutzerprofile bei den Universitäten aus *Moodle* und *Blackboard* bezogen werden.

Die COM API ermöglicht es, durch den Lernbegleiter alle Konversationen der besuchten Institutionen im Überblick zu behalten und bei Bedarf erneut Fragen zu stellen. Auf diese Weise werden soziale Kontakte in allen Lebensphasen für den Wissensaustausch im Bildungs- und Arbeitsumfeld langfristig gepflegt.

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.

Welche Aufgaben mit den Daten erfüllt werden, obliegt der clientseitigen Logik. In diesem Punkt unterscheidet sich die MI von den zuvor vorgestellten SOA-Ansätzen.

Die Interoperabilität der *Drittanbieteranwendungen* wird über die MI gewährleistet. Hierbei gilt es, für LRS, USR, COM, LCMS und CRM geeignete Standards zu ermitteln. Jede *Drittanbieteranwendung* muss innerhalb der MWA adaptiert werden, um einen entsprechenden Standard zu erfüllen. Die Tabelle 4 [S.35] zeigt die Zuordnung zwischen einer *Drittanbieteranwendung* und MWA.

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

Third Party	LRS	USR	COM	LCMS
Moodle	-	+	+	-
Blackboard	-	+	-	+
Moodle Forums	-	-	+	-
Blackboard Forums	+	-	+	-
Human Resources (HR)	-	+	-	-
Repository	-	-	-	+
Learning Locker	+	-	-	-
Piwik	+	-	-	-

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_M1: Das System fungiert als Bindeglied zwischen einer beliebigen Anzahl an *Client-* und *Drittanbieteranwendungen*. Eine Komponente des Systems muss demzufolge als *Middleware* konzipiert werden. Nach dem SOA-Paradigma wird eine *Clientanwendung* als *Service Consumer* und eine *Drittanbieteranwendung* als *Service Provider* bezeichnet.

A_M2: Das System muss den unterschiedlichen Funktionsumfang von *Drittanbieteranwendungen* berücksichtigen. Dies ist ein Problem, welches bereits durch das SOA-Paradigma gelöst wird, indem eine lose Kopplung zwischen Funktionalitäten vorgenommen wird. Zusammengehörige Funktionalitäten müssen als API zur Verfügung gestellt werden.

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.

A_M4: Eine Institution muss über das System mehrere *Drittanbieteranwendungen* auf unterschiedlichen Servern nutzen können. Die Kommunikationswege zu den *Drittanbieteranwendungen* müssen, wie durch das SOA-Paradigma definiert, als Servicebeschreibungen vorliegen. Die Kommunikationswege werden als *Route* und die Server als *Endpunkte* bezeichnet.

A_M5: Kommunikationswege müssen zentral verwaltet und von den *Clientanwendungen* (*Service Consumer*) genutzt werden können. Die zentrale Komponente wird als *Hub* bezeichnet und ist entsprechend des SOA-Paradigmas unter anderem mit einer *Service Registry* vergleichbar (Fowler et al., 2002, pp. 408-412).

A_M6: Das Hinzufügen oder Erstellen neuer Funktionalitäten darf die Stabilität und den Betrieb des Systems nicht beeinflussen. Aus diesem Grund müssen die Services als *Microservices* angeboten werden. Diese werden als MWAs bezeichnet.

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.

Bei der Konzeption der MI steht das Paradigma *Datenfreigabe durch den Nutzer* stark im Vordergrund. Drittanbieter, wie Universitäten und Arbeitsstätten, teilen untereinander keine Nutzerdaten. Eine Aggregation der Nutzerdaten sollte zentral unter der Aufsicht einer vertrauenswürdigen Kontrollinstanz erfolgen.

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

4 Systementwurf

In diesem Kapitel wird der Aufbau der auf *Routes* basierten *Middleware Infrastructure* (MI) behandelt. Zu Beginn werden die Hauptkomponenten der Abbildung 12 [S.39] erläutert.

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 MI besteht aus den Hauptkomponenten *Hub*, *Middleware API* (MWA), *Third Party Module* (TPM) und dem DAO-Manager.

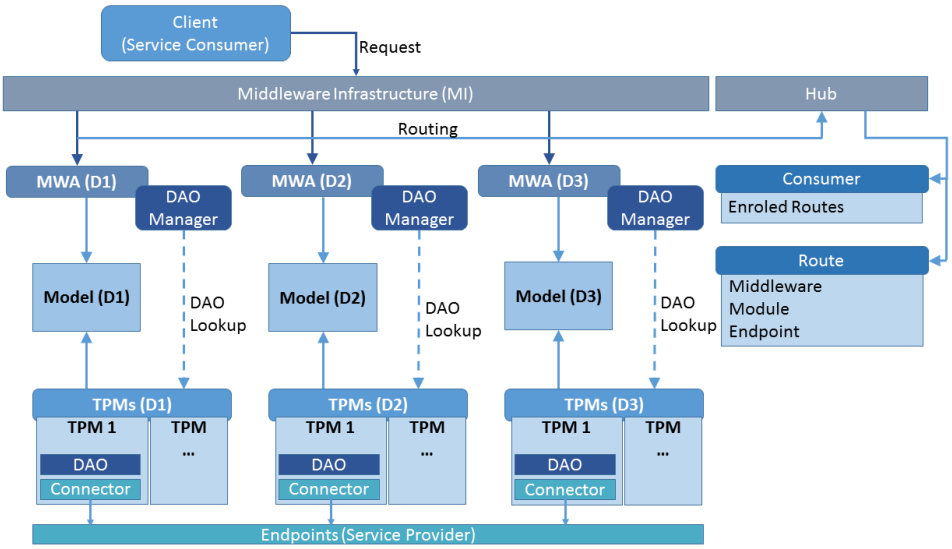

Abbildung 12: Middleware Infrastructure – MI (Eigene Darstellung)

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.

In der Abbildung 12 [S.39] ist im Vergleich zu den anderen Komponenten nur ein *Hub* abgebildet, da technisch kein zusätzlicher *Hub* benötigt wird. Die Integration weiterer *Hubs* ermöglicht verschiedene Konfigurationen, die physikalisch voneinander getrennt werden können.

4.2 Der Hub

Der *Hub* soll in der Lage sein, eine Vielzahl von Verbindungsdaten verwalten zu können. Der Einsatz einer *Middleware* kann sinnvoll sein, wenn es erforderlich ist, dass mehrere Systeme, die verschiedene Schnittstellen verwenden, miteinander kommunizieren (n:n-Relation). Ein weniger komplexer Ansatz, bei dem ein System mit mehreren Systemen kommuniziert (1:n-Relation), eignet sich ebenso für den Einsatz einer *Middleware*. Der *Hub* wurde für Anwendungsszenarien mit einer 1:n-Relation konzipiert.

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)

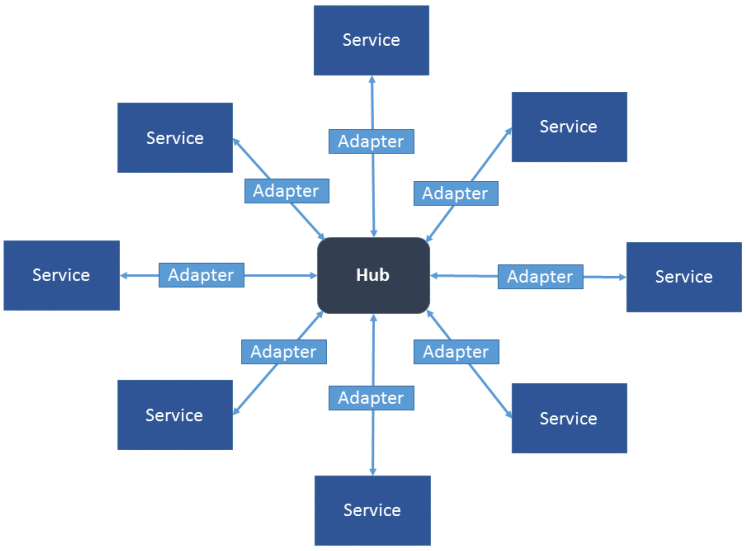

Abbildung 13: Hub and Spoke based on (Keen et al., 2004, pp. 78.)

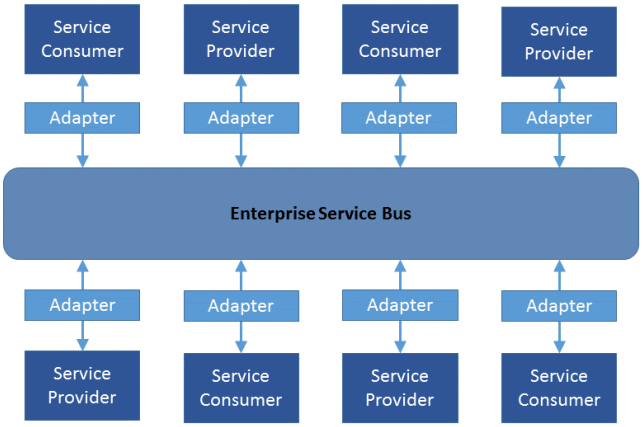

Abbildung 14: Enterprise Service Bus based on (Keen et al., 2004, pp. 77.)

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*.

Bei einem direkten Vergleich mit den SOAs aus Abbildung 13 [S.41] und Abbildung 14 [S.41] würde der aktuelle Systementwurf aufgrund der ebenfalls vorhandenen *broker technology* tendenziell der *Hub & Spoke*-Architektur zugeordnet werden.

4.2.1 Die Kommunikation zwischen Hub und Middleware API

Ein Vorteil für den *Hub* besteht darin, dass eine *Clientanwendung* nur *Routes* nutzen kann, die über den *Hub* zur Verfügung gestellt werden. Jeder *Clientanwendung* wird ein *Middleware Consumer* zugeordnet. Dieser besitzt Informationen über die Institution, die für die Entwicklung der *Clientanwendung* verantwortlich ist. Auf diese Weise können die Zugangsberechtigungen der *Routes* sichergestellt werden. Daraus folgt, dass nicht jede *Clientanwendung* alle *Routes* des *Hubs* nutzen kann.

Die Abbildung 15 [S.42] verdeutlicht, wie der *Hub* innerhalb der MI kommuniziert.

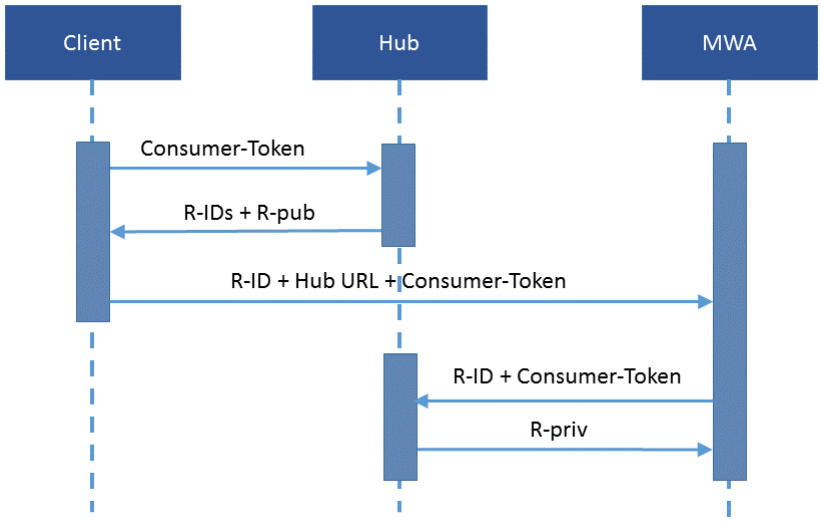

Abbildung 15: Hub Communication (Eigene Darstellung)

Eine *Clientanwendung* übermittelt den *Consumer Token* an den Hub. Anhand dessen kann der *Hub* den *Middleware Consumer* eindeutig zuordnen und ermitteln, welche *Routes* autorisiert sind. Anstatt der *Clientanwendung* alle verfügbaren *Routes* mitzuteilen, werden nur einige ausgewählte öffentliche Parameter (R-pub) übermittelt sowie die für die MWA benötigten *Route-IDs* (R-IDs). Viele Daten, die der *Hub* in Form einer *Route* persistiert, sind vertraulich und sollten nicht an die *Clientanwendung* übermittelt werden. Bei einer Clientanfrage an die MWA wird durch eine *Route-ID* (R-ID) der *Endpunkt* und die zu verwendende Anwendung eindeutig identifiziert.

Neben der *Route-ID* übermittelt die *Clientanwendung* zusätzlich die URL des Hubs. Somit erkennt die MWA den *Hub*, der für die Authentifizierung des *Middleware Consumers* verwendet wurde. Des Weiteren wird der Consumer-Token an die MWA übertragen. Technisch ist der *Consumer Token* nicht erforderlich, um eine Verbindung mit einem *Endpunkt* herzustellen. Ohne den Token wäre es jedoch jeder *Clientanwendung* möglich, mit einer bekannten R-ID eine *Route* zu nutzen, selbst wenn diese für den Consumer nicht vorgesehen wurde.

Sobald die Clientanfrage in der MWA eingegangen ist, stellt diese eine Anfrage an den *Hub*. Durch diesen Prozessschritt werden die privaten *Route*-Daten (R-priv) abgefragt. Mithilfe der R-priv-Daten wird eine Verbindung mit einem *Endpunkt* hergestellt und ein TPM mit geeigneter *Geschäftslogik* gefunden.

Der Austausch von R-priv-Daten zwischen *Hub* und MWA ist ohne geeignete Authentifizierung eine Sicherheitslücke. Ziel ist es zu verhindern, dass in einem verteilten System unbefugte Parteien oder unbekannte MWAs an R-priv-Daten gelangen. Aus diesem Grund besitzt jede MWA in der Konfiguration einen API Key. Die MWAs werden bei dem *Hub*, der die benötigten *Routes* verwaltet, registriert. Sowohl die MWA als auch der *Hub* besitzen den *API Key*. Im aktuellen Stand der Entwicklung wird prototypisch nur der *API Key* überprüft. Begünstigt und vorgesehen ist ein symmetrisches Verschlüsselungsverfahren.

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.

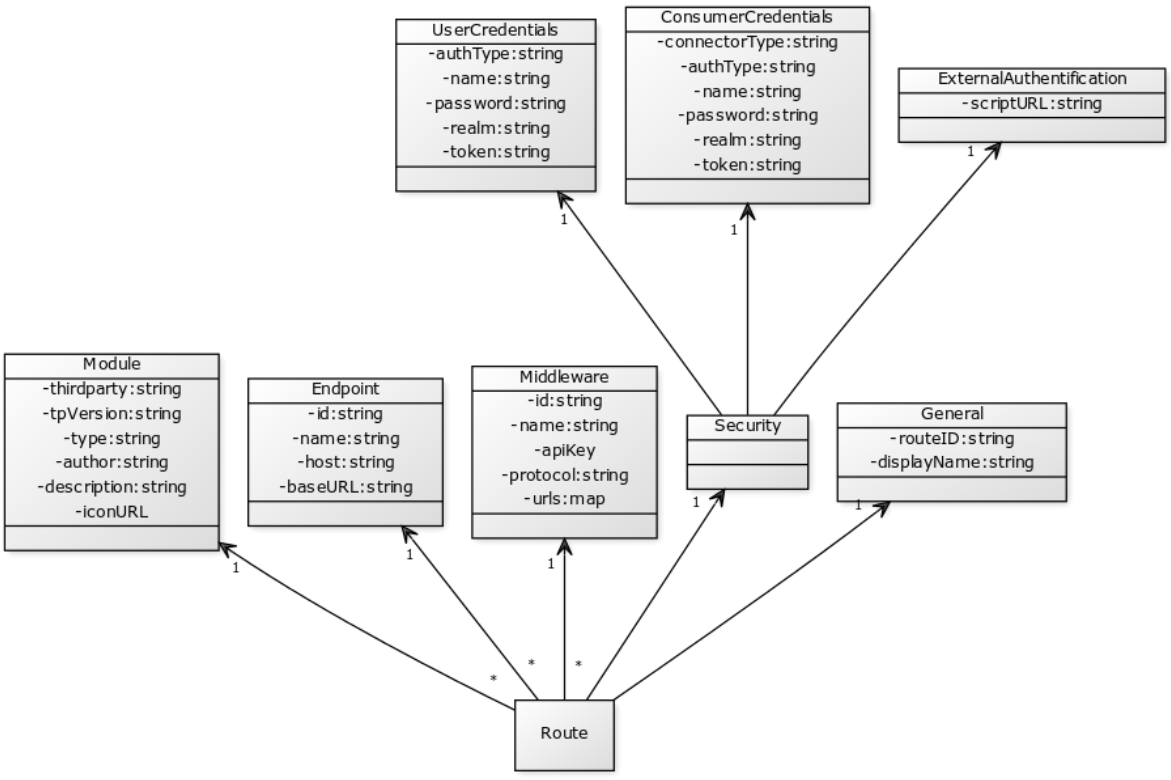

Abbildung 16: Route (Eigene Darstellung)

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 `ExternalAuthentification`, `ConsumerCredentials` und `UserCredentials`.

Die Klasse `ExternalAuthentification` 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 `UserCredential` ist vergleichbar mit der Klasse `ConsumerCredential`. Ein Unterschied besteht darin, dass die Nutzerdaten nicht im *Hub* persistiert werden. Ein Objekt dieser Klasse wird erst bei einer Clientanfrage initialisiert.

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.

Eine Anbindung an einen *Hub* erfolgt bei jeder Nutzung erneut. Jeder *Hub* hat einen Management-Key. Ohne diesen ist es der Webapplikation nicht möglich, die *Management Services* eines Hubs ausführen zu können.

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.

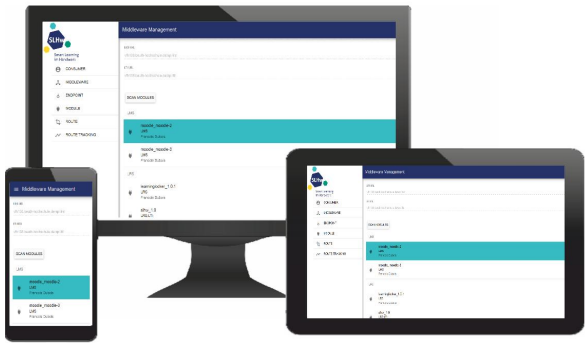

Abbildung 17: Showcase Hub Management (Eigene Darstellung)

4.3 Die Middleware API (MWA)

Die Middleware API (MWA) hat die Aufgabe, zu verschiedenen Services einer *Domäne* Schnittstellen bereitzustellen. Hierbei ist anzumerken, dass alle Datenmodelle der jeweiligen Services abstrakt konzipiert werden sollten. Begründen lässt sich dies mit der Heterogenität von *Backend*-Systemen. Jedes TPM einer *Domäne* muss kompatibel mit den entsprechenden Schnittstellen beziehungsweise den standardisierten Datenmodellen der MWA sein. Das Kernproblem dabei ist, die Funktionalitäten zu ermitteln, die ein elementarer Bestandteil einer *Domäne* sind.

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.

Voraussetzung für die Dynamisierung der *Geschäftslogik* ist das sogenannte Routing. Eine *Route* definiert, wie bereits erläutert, den Verbindungsweg von der MWA bis zum gewünschten *Endpunkt*. Eine MWA besitzt vorerst nur die Logik zum Aufruf eines TPM sowie die standardisierten Datenmodelle. Die Kommunikation mit einer *Clientanwendung* erfolgt über das *Hypertext Transfer Protocol*-Protokoll (HTTP).

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

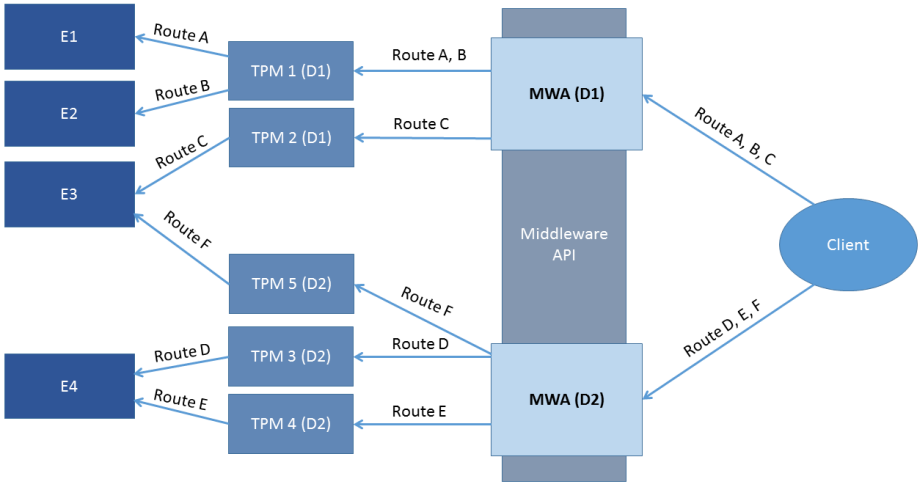

Abbildung 18: Middleware Routing (Eigene Darstellung)

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.

Die Abbildung 19 [S.47] verdeutlicht die Consumer- und User-Authentifizierung.

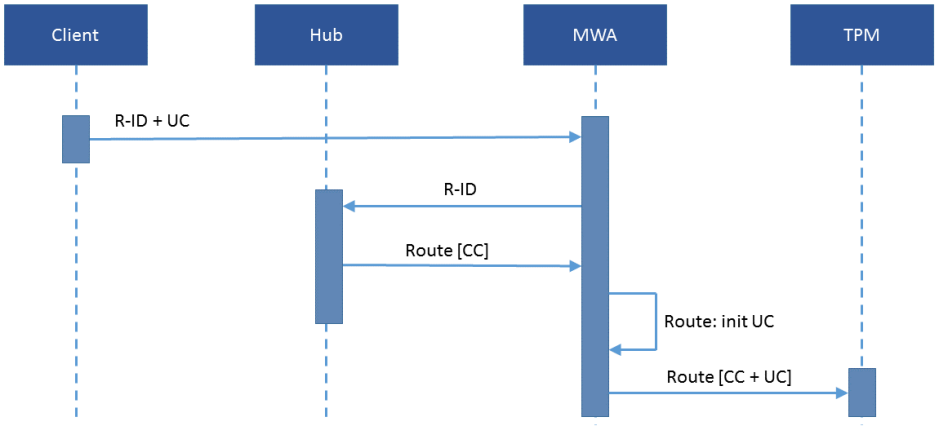

Abbildung 19: Consumer & User Authentication (Eigene Darstellung)

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

Durch das serverseitige Persistieren von CC besteht die Möglichkeit, *Routes* zu individualisieren. Eine *Route* wird an einen *Middleware Consumer* gebunden und führt alle Anfragen repräsentativ aus.

Die Abbildung 20 [S.48] beschreibt anhand eines abstrakten Szenarios, wie verschiedene gesicherte Sektionen einer *Drittanbieteranwendung* durch die Verwendung von *Routes* identifiziert werden könnten.

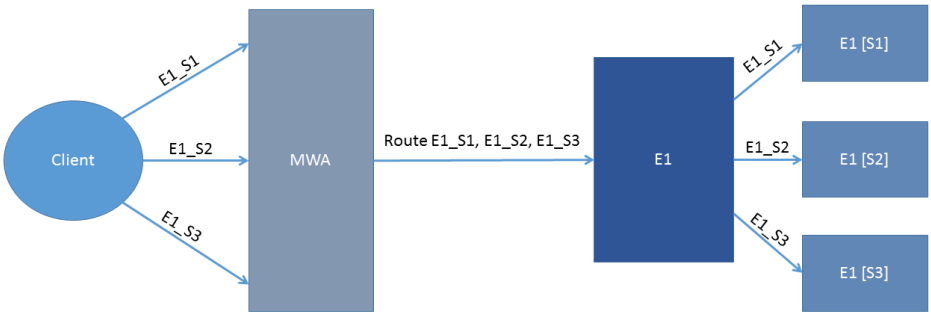

Abbildung 20: Individualized Routes (Eigene Darstellung)

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 [S.43]), das es ermöglicht, ein Script anzufordern. Dieses Script besitzt die Logik zur Authentifizierung. Es ist in Betracht zu ziehen, ausschließlich 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 [S.43]) sowie den Namen und das Passwort eines Nutzers. Der Rückgabewert ist bei einem Erfolg ein User-Token der Dienstleistung.

Die Abbildung 21 [S.49] beschreibt die Bypass-Authentifizierung in Prozessschritten.

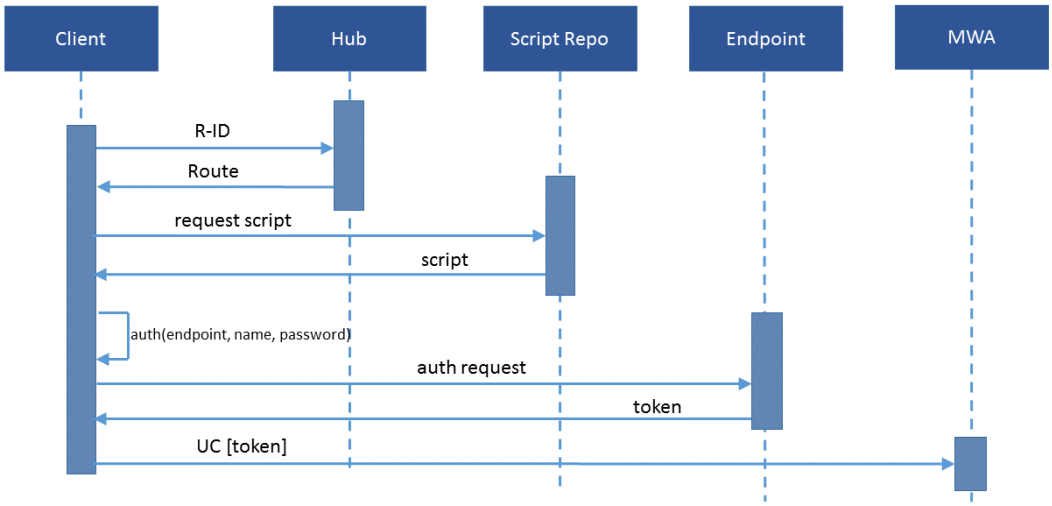

Abbildung 21: Bypass Authentication (Eigene Darstellung)

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.

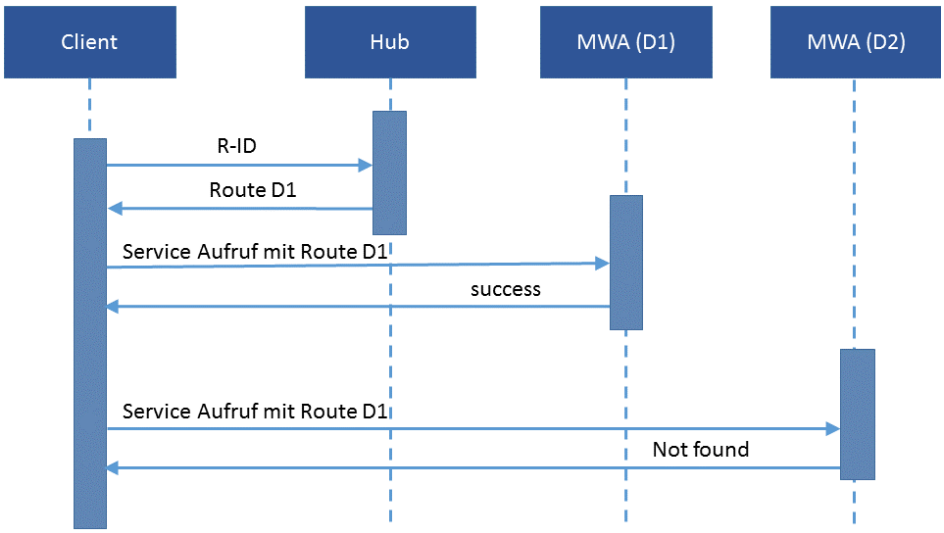

Abbildung 22: Route Domain (Eigene Darstellung)

4.4 Der DAO Manager als Service Broker

Der *DAO Manager* ist eine relevante Komponente in der MI. Die Aufgabe des *DAO Managers* ist es, zu einem aufgerufenen Service ein passendes *Data Access Object* (DAO) (Alur et al., 2001, pp. 371-374) zu finden. Ein DAO besitzt die nötigen *Create-, Read-, Update- und Delete- Methoden* (CRUD). Der *DAO Manager* delegiert an die passende Methode. Das Delegieren von Logik ist ein bekanntes *Java-EE-Entwurfsmuster* und ist unter der Bezeichnung *business delegate* bekannt (Stal, 2015; Alur et al., 2001, pp. 108-111). Der *DAO Manager* fragt zudem die für das Auffinden eines DAO (*lookup*) benötigten privaten *Route*-Daten (R-priv) von dem *Hub* an.

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:

Tabelle 5: REST Mapping (Eigene Tabelle)

HTTP Method	DAO (CRUD)
POST	Create-Methode
GET	Read-Methode
PUT	Update-Methode
DELETE	Delete-Methode

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.

Die MI beschränkt sich nicht auf einzelne MWAs. Für jede *Domäne* lässt sich eine MWA an einen *Hub* anbinden. Eine MWA kann Nutzer an verschiedenen *Hubs* authentifizieren. Die *Clientanwendung* übermittelt hierbei den zuständigen *Hub*. Eine domänenspezifische MWA kann mehrere Instanzen bereitstellen, die über einen *Load Balancer* (Proxy) angesteuert werden können.

Es können mehrere *Hubs* existieren. Ein *Hub* verwaltet *Routes* und *Middleware Consumer*. Aus diesem Grund ist ein *Hub* unter anderem mit einem Authentifizierungsserver vergleichbar.

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 möglicher Lösungsansatz ist das lokale Zwischenspeichern von *Routes* in der MWA. Bei einer Clientanfrage wird nur initial die *Route* von dem *Hub* angefragt. Es ist zu beachten, dass der *Consumer Token* mit einer *Route* verknüpft werden sollte. Auf diese Weise ist auch eine lokale Authentifizierung weiterhin möglich.

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*).

Die Abbildung 23 [S.53] zeigt ein verteiltes System, um die Möglichkeiten der Lastenverteilung (engl. *load balancing*) zu verdeutlichen.

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.

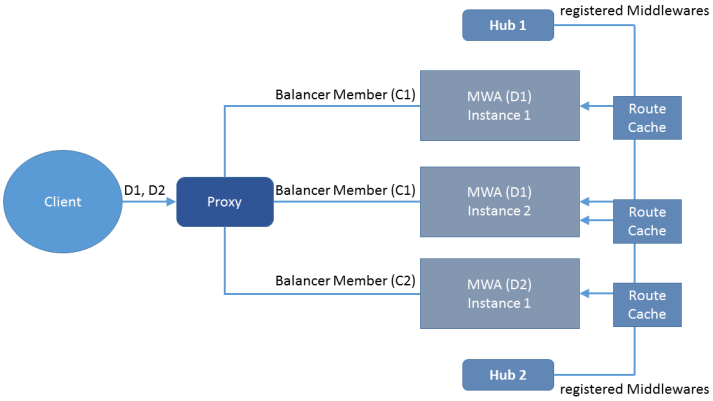

Abbildung 23: Middleware Balancing (Eigene Darstellung)

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)

Die verwendeten Dienstleistungskomponenten werden in MWAs überführt. Entscheidend sind hierbei die für die einzelnen Services verwendeten Entities (Datenmodelle). Eine Möglichkeit wäre es, bereits etablierte Standards zu finden, die alle Funktionalitäten einer Komponente abbilden. Dies kann eine gute Grundlage für ein abstraktes *Entity* sein und die Interoperabilität begünstigen. Für die Komponenten Tracking und LMS wurden in Tabelle 6 [S.53] folgende Standards ermittelt:

Tabelle 6: API Standards (Eigene Tabelle)

API	Standard
Tracking (LRS)	ExperienceAPI (xAPI) (TinCan, n.d.)
LMS	IMS CC (Course) (IMS Global Learning Consortium, 2015a)

Nach der Erstellung der MWAs ist in Betracht zu ziehen, die ersten TPMs zu implementieren. Die Anwendungen mit dem höchsten Marktanteil sollten priorisiert werden. Es ist nicht das Ziel, jedes System zu integrieren. Der Mehrwert der MI ist die Verwendung verschiedener Dienstleistungen für mehrere *Middleware Consumers*. Diese können in der E-Learning-Branche zum Beispiel Bildungseinrichtungen sein.

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.

Abschließend wird die Abbildung 12 [S.39] aus der Einführung des Kapitels *Systementwurf* in der Abbildung 24 [S.54] zu einer MI für die E-Learning-Branche überführt.

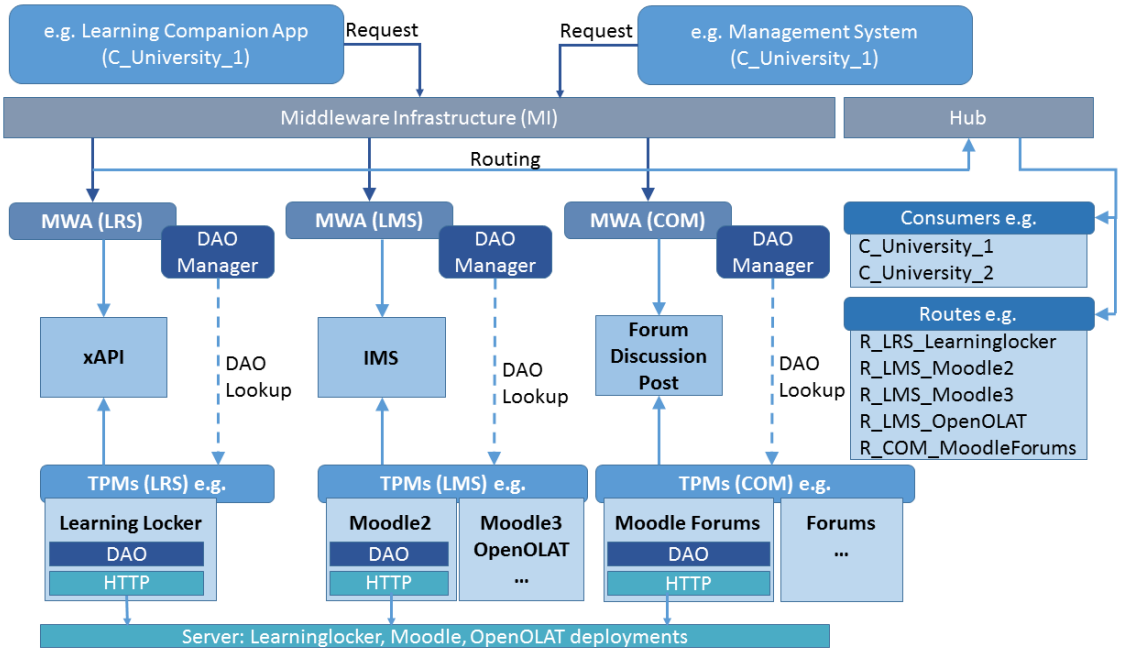

Abbildung 24: Middleware Infrastructure - E-Learning (Eigene Darstellung)

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).

Vielmehr setzt die MI klare API-Standards voraus. Eine *Clientanwendung* kommuniziert mit der MI über eine MWA. Die *Clientanwendung* kann der MI mitteilen, welche Drittanbieterdienstleistung genutzt werden soll (1:n-Relation). Dies wird durch *Routes* gewährleistet. Neben den *User Credentials* (UC), die das Paradigma „Datenfreigabe durch den Nutzer“ ermöglichen, ist über *Consumer Credentials* (CC) eine Authentifizierung der Dienstleistungsnehmer möglich.

Durch die Zentralisierung des *Hubs* besteht bei zu vielen Anfragen das Risiko eines *Bottlenecks*. Aus diesem Grund wird ein Konzept für das Zwischenspeichern von *Routes* in der MWA entworfen. Für den Fall, dass kein *Single-Sign-On* (SSO) über einen Authentifizierungsserver vorgesehen ist, wird durch eine spezielle Bypass-Authentifizierung eine direkte Kommunikation zwischen *Drittanbieteranwendung* und *Clientanwendung* gewährleistet. Eine MWA erhält in diesem Szenario keine Zugangsdaten eines Nutzers.

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).

Nach dem IoC wird der benutzerspezifische Programmcode registriert und von der MI verwaltet. Durch die eindeutigen Schnittstellen sind die Aufgaben beziehungsweise Entwicklungsschritte klar definiert. Die Einarbeitungs- und die Entwicklungszeit werden erheblich reduziert. Der benutzerspezifische Programmcode ist auf die problemlösungsorientierten Codefragmente zu reduzieren. Der Kontrollfluss wird vollständig durch die funktionalen Module abgedeckt.

Die Abbildung 25 [S.57] bietet eine Übersicht der funktionalen Module. Zudem wird die Integration domänenspezifischer Lösungen miteinbezogen.

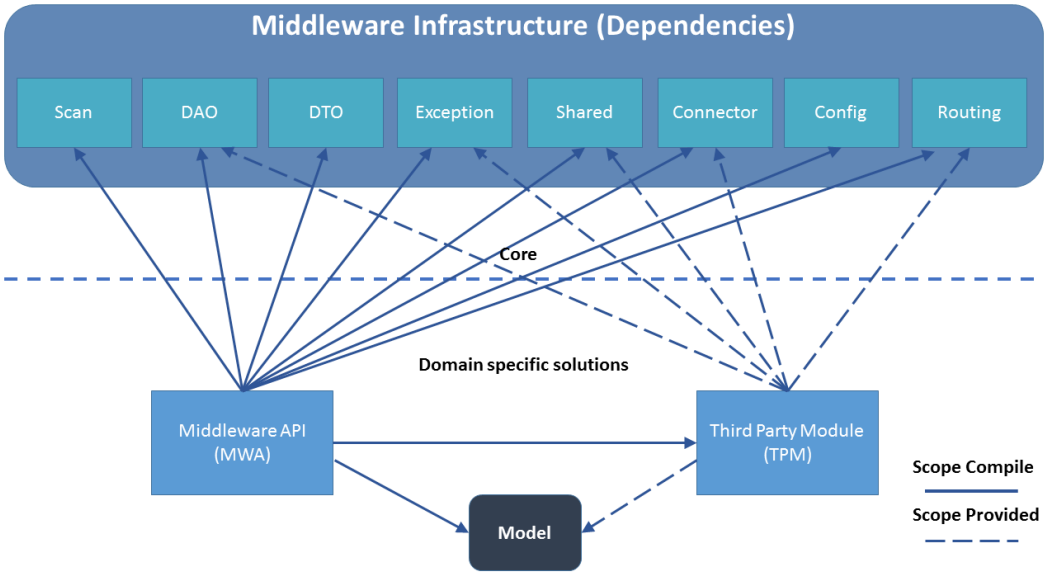

Abbildung 25: Module integration dependencies (Eigene Darstellung)

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.

Alle Schnittstellen orientieren sich an den Datenbankoperationen Erstellen (*Create*), Lesen (*Read*), Aktualisieren (*Update*) und Löschen (*Delete*). Diese Operationen sind unter dem Akronym CRUD bekannt.

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.

```
1 public abstract class AbstractRouteDAO<E> {
2         public abstract E create(Route route, E entity, Map<String,
            Object> optionalParams);
3
4         public abstract List<E> read(Route route, E entity, Map<String,
            Object> optionalParams);
5
6         public abstract E update(Route route, E entity, Map<String,
            Object>  optionalParams);
7
8         public abstract boolean delete(Route route, E entity, Map<String,
             Object> optionalParams);
9 }
```

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.

Der dritte Parameter übergibt optionale Parameter. Diese können Anweisungen der MWA sein. Häufig werden Path-, Query- oder Header-Parameter übergeben.

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.

Ein DAO wird als zustandsloses Enterprise Java Bean (EJB, Oracle, 2013c) aufgerufen. Bei einem zustandslosen EJB werden alle Variablen für einen Request neu initialisiert. Ein *Stateless* EJB wird zur Laufzeit über Code-*Injektion* eingebunden. Die JNDI wird im Normalfall statisch definiert.

Erst während der Clientanfrage ist das benötigte DAO bekannt. Eine Möglichkeit ist es, alle potenziellen DAOs über die jeweiligen JNDIs einzubinden. Ein statisches Einbinden über Annotationen ist hinsichtlich der Wartbarkeit sehr aufwendig. Es soll vermieden werden, dass der Code der MWA beim Hinzufügen eines TPM zu verändern ist.

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.

```
1  public <T> T lookup(String jndi) {
2          Context context;
3          try {
4                  context = new InitialContext();
5                  Object obj = context.lookup(jndi);
6                  return (T) obj;
7          } catch (NamingException e) {
8                  logger.error("EJB not found for JNDI -> " + jndi);
9                  throw new CustomException("EJB not found","500",this.
                     getClass());
10         }
11 }
```

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

In der Zeile 4 wird ein neuer `InitialContext` instanziiert. 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]).

```
1  public static final String getJNDI(final Route route, final Class<?>
     entity) {
2          String thirdparty = route.getModule().getThirdparty();
3          String version = route.getModule().getTpVersion();
4          Security security = route.getSecurity();
5          ConsumerCredentials cc = security.getConsumerCredentials();
6          ConnectorType connectorType = cc.getConnectorType();
7
8          return JNDI_MODULE_PREFIX + thirdparty + "_" + version + "_" +
             entity.getSimpleName() + "_" + connectorType.name();
9  }
```

Listing 3: Dynamic JNDI (Eigenes Listing)

In der Zeile 8 wird eine dynamische JNDI auf Basis der *Route* und dem *Entity* erstellt. Die Konstante `JNDI_MODULE_PREFIX` ist in dem Listing 3 [S.59] nicht abgebildet. In diesem Fall wird das Prefix `java:module/` verwendet. Daraus folgt, dass die EJBs innerhalb des *Wildfly Modules* zu finden sind.

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.

```
1  @EJB
2  private DAOManager daoManager;
3  public ContainerResponse<Course> getCourses(/* API Params */) {
4          // create dynamic JNDI based on Course entity and Route
5          String jndi = DAOManager.getJNDI(route, Course.class);
6
7          // find a potential DAO with given JNDI
8          AbstractRouteDAO<Course> courseDAO = daoManager.lookup(jndi);
9
10         // call specified READ method on courseDAO delegate
11     return daoManager.callMethod(route, courseDAO, Action.READ, null,
         optParams);
12 }
```

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.

Ein Filter wird nach Vorgabe des `Filter` Interface implementiert. Anschließend wird über eine *Filter Chain* festgelegt, zu welchem Zeitpunkt dieser aufgerufen wird. Der Listing 5 [S.61] zeigt das `Filter` Interface sowie eine angedeutete *Filter Chain*.

```
1  public interface Filter {
2      public void filter(Route route, Object entity, Map<String, Object>
         optionalParams, Action action);
3  }
4
5  @LocalBean
6  @Stateless(name = JNDI.COURSE_HTTP_DAO)
7  public class CourseDAO extends AbstractRouteDAO<Course> {
8
9      @Override
10     @PreFilters(SecurityFilter.class, Filter2.class)
11     @PostFilters(Filter3.class, Filter4.class)
12     public List<Course> read(Route route, Course course, Map<String,
         Object> optionalParams) {
13         // some code ...
14     }
15 }
```

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)

Der Begriff *Data Transfer Object* (DTO, Fowler et al., 2002, pp. 347-356) steht für ein Entwurfsmuster, bei dem Daten gebündelt über ein Netzwerk übertragen werden. In dieser Arbeit wurde ein funktionales Modul integriert, das für ausgehende Daten (Response) ein `ContainerResponse` DTO zur Verfügung stellt.

In dem Listing 6 [S.62] wird ein JSON-Objekt einer `ContainerResponse` dargestellt.

```
{
   "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.

Die `ContainerResponse` wird in der Regel von dem `DAOManager` initialisiert. Rückgabewerte aus einem DAO werden in dem Attribut `data` übermittelt. Besteht der Bedarf, eine eigene `ContainerResponse` in einem Service zu nutzen, wird diese durch das Entwurfsmuster *Builder* (Gamma et al., 2011, pp. 119-130) erstellt, wie das Listing 7 [S.62] darstellt.

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

Listing 7: ContainerResponse - Builder Pattern (Eigenes Listing)

Neben der `ContainerResponse` existiert ein weiteres DTO-Format. Wenn eine MWA eine *Route* von dem *Hub* anfragt, wird die Aktivität in dem *Hub* mithilfe des DTO `Record` aufgezeichnet. Dieses DTO wird von dem `HubManager` in dem funktionalen Modul *Routing* verwendet.

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 Verbindungsprotokolle sehr verschieden sein können. Der erste und bisher einzige `Connector` ist der `HTTPConnector`. Dieser wird zum Aufrufen von *Webservices* (vorzugsweise REST) genutzt.

Die Implementierung des `HTTPConnector` wird durch das *RestEasy Client Proxy Framework* 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 Voraussetzung 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-Variablen mit der `@Inject` und der erzeugten Annotation `@ApplicationProperty(propertyName)` zu injizieren, wie das Listing 8 [S.64] verdeutlicht.

```
1 @Inject
2 @ApplicationProperty(name = "mw_apiKey")
3 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.

Neben dem Anfragen von *Routes* verwendet der `HubManager` einen `HubStorage`. Dieser hat die Aufgabe, eingehende *Routes* lokal in einem Zwischenspeicher zu halten, wie das Listing 9 [S.65] zeigt.

```
1  @Lock(LockType.WRITE)
2  public void addRoute(String hubURL, String consumerToken, Route route) {
3          String host = getHost(hubURL);
4          if(!hubStorage.containsKey(host)) {
5                  hubStorage.put(host, new HashMap<String, Route>());
6          }
7
8          String key = getKey(consumerToken, route.getId());
9          hubStorage.get(host).put(key, route);
10 }
11
12 public String getKey(String consumerToken, String routeID) {
13         return Base64.encodeBytes((consumerToken + ":" + routeID).
             getBytes());
14 }
```

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.

```
1  public ContainerResponse<?> scan() {
2          ClassLoader cl = Thread.currentThread().getContextClassLoader();
3          try {
4                  // searching for module.properties files
5                Enumeration<URL> urls = cl.getResources(REFLECTION_MARKER);
6                while (urls.hasMoreElements()) {
7                      URL url = (URL) urls.nextElement();
8                      // create virtual file from resource url
9                      VirtualFile vf = asVirtualFile(url);
10                     // get required physical path
11                     File file = vf.getPhysicalFile();
12                     // read module properties from physical file
13                     readModule(file);
14               }
15         } catch (IOException e) { /* exception handling */ }
16         // return all available module descriptions
17         return ContainerResponse.ok().entity(modules).build();
18 }
```

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.

Ein TPM funktioniert theoretisch ohne die besagte Datei. Problematisch ist jedoch das Registrieren an den *Hub*. Dieser könnte keine Informationen von dem verfügbaren TPM anfragen und somit keine vollständige *Route* erstellen.

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).

Der benötigte physikalische Dateipfad wird über eine *Virtual File* (VF) generiert. In dem Listing 11 [S.67] wird aus einer Ressourcen-URL aus dem `ClassLoader` der MWA in Zeile 7 eine VF erzeugt.

```
1  private VirtualFile asVirtualFile(URL url) {
2          // check if the given url is type of virtual file
3          if ("vfs".equals(url.getProtocol())) {
4                  try {
5                          URLConnection conn = url.openConnection();
6                          // extract virtual file from url content
7                          VirtualFile virtualFile = (VirtualFile)conn.
                             getContent();
8                          return virtualFile;
9                  } catch (IOException e) {
10                         e.printStackTrace();
11                 }
12         }
13         return null;
14 }
```

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]).

Bei den Service-Parametern sollte ein kontextueller Bezug bestehen. Zu spezifische Parameter überfüllen die Schnittstellen und können nicht von allen TPMs erfüllt werden. Attribute sollten immer als `String` typisiert werden. Es ist nicht absehbar, welche Datentypen von einer *Drittanbieteranwendung* genutzt werden.

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 COM API versucht, verschiedene Forenimplementierungen zu vereinheitlichen. Es gibt keinen offiziellen Standard. Die Komplexität der Datenmodelle kann minimal auf `Forum`, `Discussion` und `Post` reduziert werden. Es folgt in Abbildung 26 [S.69] ein Klassendiagramm mit den für die Funktionalität eines Forums benötigten Attributen.

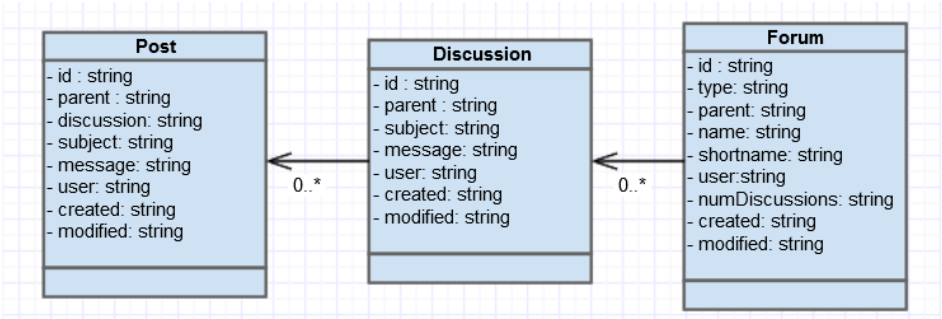

Abbildung 26: Forum (Eigene Darstellung)

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 von Attributen 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 minimalistische *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*.

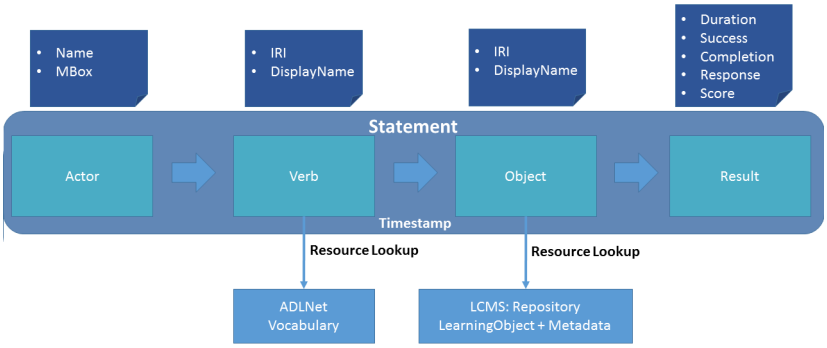

Abbildung 27: xAPI Statement structure (Eigene Darstellung)

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 Addresse) durch eine USR API identifiziert werden.

Die LMS API

Die LMS MWA wurde bisher nicht ausreichend konsolidiert. In den vorherigen Kapiteln wurde stets eine Trennung zwischen LMS und LCMS sichergestellt. Aufgrund der zeitlichen Begrenzung dieser Arbeit erfolgt keine eindeutige Trennung.

Die wichtigste Aufgabe der LMS API ist das Bereitstellen von Kursstrukturen und Lerninhalten (Lernobjekte). Hierbei handelt es sich um LCMS-Dienstleistungen. Aktuell werden zusätzlich LMS-Dienstleistungen, wie Nutzerprofile (*vCard*) und Kurs-Enrolments, bereitgestellt.

Für das Bereitstellen von Kursstrukturen wurde der IMS *Common Cartridge* Standard (IMS CC, IMS Global Learning Consortium, 2015a) gewählt. Dieser wird eingesetzt, um interoperabel Kursstrukturen in LMS importieren und exportieren zu können und wurde für eine Archivierung optimiert. Die Daten werden mit *Extensible Markup Language* (XML) beschrieben und über Dateistrukturen referenziert. In der Regel wird der Datenbestand in einer zip-Datei verpackt.

Es wird angestrebt, den IMS-CC-Standard mit JSON zu beschreiben und eine Alternative zum Referenzieren von Dateien zu gewährleisten. Auf diese Weise können sowohl Repositories als auch Live-Kurse eines LMS über die Kurs-Schnittstelle adaptiert und angefragt werden.

Grundlegend besteht das IMS-CC-Datenmodell aus `Metadata`, `Organizations` und `Resources`. Die Metadaten liefern Informationen zu dem Kurs. Die Organization beschreibt den Aufbau (Baumstruktur) des Kurses. Die Ressourcen referenzieren Beschreibungen von Lerninhalten, die nach dem IMS-CC-Standard in einer separaten XML-Datei abgelegt werden.

Bei der Anpassung des Standards bleiben die Klassen `Metadata` und `Organizations`, abgesehen von der Umwandlung von XML zu JSON, unverändert. Bei der Klasse `Resources` wird die ursprüngliche `List` durch eine `HashMap` ersetzt. Der Key ist der Identifikator einer `Resource` (`identifierref`). Auf diese Weise wird der *lookup* einer Resource mit hoher Wahrscheinlichkeit auf die Komplexität O(1) reduziert. Das Objekt `Value` ist die Beschreibung eines Lerninhalts. Es wird auf keine XML-Datei referenziert. Zudem wird kein Dateisystem benötigt.

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

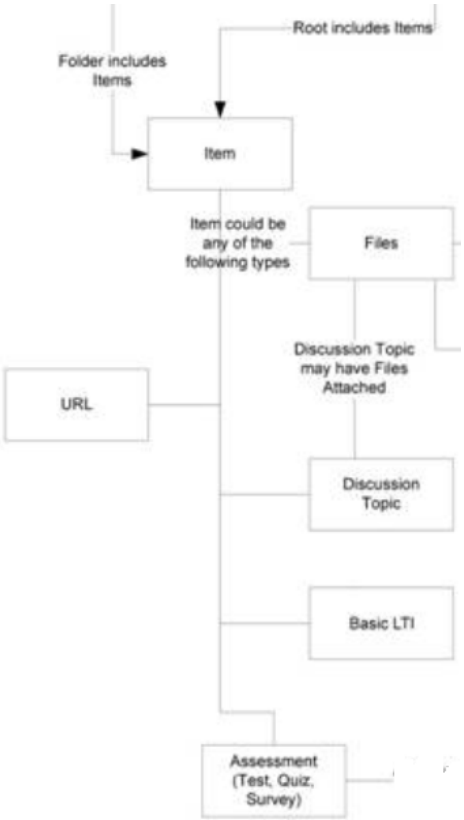

Abbildung 28: Part of IMS CC specification (IMS Global Learning Consortium, 2015a)

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 [S.73] aufzeigt:

Tabelle 7: Hub API mappings (Eigene Tabelle)

Services	Management API	Route API	Public API
Auth	-	-	+
Config	+	-	-
Route	+	-	-
Middleware	+	-	-
Endpoint	+	-	-
Middleware Notification	+	-	-
Tracking	+	-	-
Consumer	+	-	-
RouteInfo	-	+	-

Die Management-API stellt alle Operationen zum Erstellen und Verwalten von *Routes* und *Consumers* bereit. Diese Schnittstelle wird dazu verwendet, um Verwaltungsoberflächen zu erstellen. Für diese Arbeit wurde ein auf *AngularJS* basierendes responsives

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.

Die Public-API besitzt einen `AuthService`, über den ein *Middleware Consumer* alle autorisierten *Routes* anfragen kann. Zu beachten ist, dass nur die nötigsten Informationen einer *Route* und keine sicherheitsrelevanten Daten übermittelt werden.

5.3 Zusammenfassung

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

Entsprechend dieser Vorgabe wurden viele funktionale Module für eine *Inversion of Control* (IoC)-Nutzung optimiert. Auf diese Weise schreiben Entwickler einen problemlösungsorientierten Programmcode. Der eigentliche Kontrollfluss beziehungsweise die Standardabläufe in der MI werden übergeordnet verarbeitet. Die Einarbeitungs- und Entwicklungszeit kann durch IoC reduziert werden.

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

Tabelle 8: Functional modules (Eigene Tabelle)

Funktionales Modul	Beschreibung
Scan	Auflösen von verfügbaren Third Party Modules in der Middleware API
DAO	Aufsuchen von Drittanbieterschnittstellen nach CRUD-Operationen
DTO	Datentransferobjekte für einen einheitlichen Datenaustausch
Exception	zentrale Fehlerbehandlung in der Middleware Infrastructure
Shared	Util-Klassen und zwischen funktionalen Modulen geteilte Datenmodelle
Connector	Verbindungstypen zu Drittanbieteranwendungen für Third Party Modules
Config	Einlesen einer Konfigurationsdatei für die Middleware Infrastructure
Routing	Anfragen von Routen

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.

Die Public-API authentifiziert *Middleware Consumer* und übermittelt verfügbare *Routes*. Die Management-API ermöglicht das Verwalten von *Consumer* und *Routes* über eine Benutzeroberfläche. Die Route-API liefert alle Daten einer *Route* an die MWA, um eine Verbindung mit einer *Drittanbieteranwendung* aufbauen zu können.

6 Evaluation

In diesem Kapitel wird die konzipierte *Middleware Infrastructure* (MI) anhand von Softwarekriterien beurteilt. Zum einen wird durch die qualitative Evaluation der Ist-Stand der MI in dem SLHw-Projekt betrachtet, zum anderen ermittelt die quantitative Evaluation durch Messungen (Monitoring) der MWAs einen Richtwert für die Performance. Die Ergebnisse der Evaluation sollen ermitteln, ob die Anforderungen für einen produktiven Einsatz oder eine Nachnutzung erfüllt werden. Zudem sind noch fehlende Funktionalitäten für eine Weiterentwicklung zu determinieren.

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*.

Evaluiert 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)

In dem Projekt *Smart Learning – Medieneinsatz in der handwerklichen Weiterbildung* (SLHw) – wurde ein Kurs begleitend zur staatlich anerkannten Fortbildung zum Gebäudeenergieberater (GEB) angeboten und darin Videos, Animationen, Bilder, Texte und interaktive Übungen bereitgestellt. Dieser Kurs mit 8 Teilnehmern wurde für einen Zeitraum von 24 Wochen konzipiert und von der *Handwerkskammer Berlin* durch ein LMS (*Moodle* 2) realisiert. Die Kursinhalte richteten sich an Handwerker, Architekten und Ingenieure.

Advanced Web Technologies Kurs (TU Berlin / Fraunhofer FOKUS)

Der Kurs *Advanced Web Technologies* (AWT) mit 154 eingeschriebenen Studenten von der *Technischen Universität Berlin* wurde über einen Zeitraum von 22 Wochen angeboten. Das *Fraunhofer FOKUS* realisierte diesen Kurs durch ein LMS (*Moodle* 3). Es wurden überwiegend Folien bereitgestellt, um das Wissen über relevante Web-Technologien sowie Ausblicke über zukünftige Entwicklungen, beispielsweise in Forschungsprojekten und Standardisierungen, zu geben.

JavaFX Kurs (Beuth Hochschule für Technik Berlin)

Der Test-Kurs *Java-FX* mit 53 eingeschriebenen Studenten wurde über die *Beuth Hochschule für Technik Berlin* für Online-Studenten angeboten und von dieser durch ein LMS (*Moodle* 2) realisiert. Über einen Zeitraum von 10 Wochen vermittelte dieser Kurs Grundlagen zur Erstellung von grafischen Benutzeroberflächen mithilfe des *JavaFX-Frameworks*. Es wurden Codeausschnitte, Dateien mit Programmcode, Bilder, Texte und interaktive Übungen bereitgestellt.

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)

Ergänzend zu der digitalen Infrastruktur wird eine *Lernbegleiter-App* (engl. Learning Companion App) entwickelt und diese auf der Ebene der Präsentationsschicht angesiedelt. Die Aufgabe der *Lernbegleiter-App* ist, alle Inhalte von Dienstleistungen der digitalen Infrastruktur darzustellen. Die *Lernbegleiter-App* ist eine Webanwendung und kann auf allen gängigen Endgeräten wie Desktops, Tablets und Smartphones verwendet werden. Lerner erhalten über die *Lernbegleiter-App* geeignete Empfehlungen zu Lernobjekten. Zu diesem Zweck wurde eine *Recommendation Engine* entwickelt. (Krauss et al., 2016a)

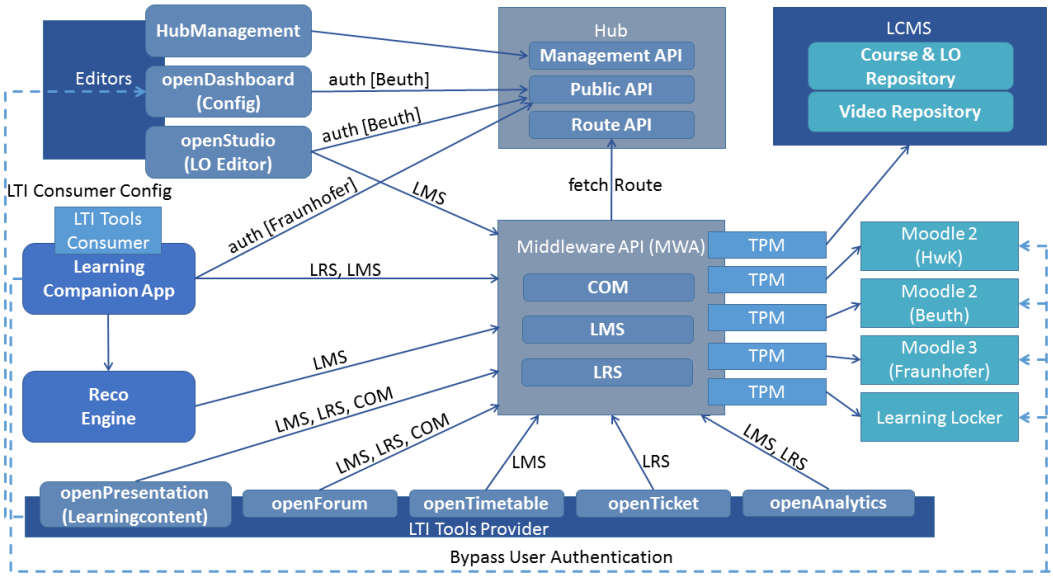

Abbildung 29: SLHw openLCMSI (Eigene Darstellung)

Für das Aufsetzen der MI sind Grundkenntnisse der Java-EE-Entwicklung im Zusammenhang mit dem Application Server *Wildfly* von Vorteil. Für das Verwalten von *Routes* und *Consumer* wird eine *MongoDB* vorausgesetzt. Zudem wird mindestens ein weiterer Server mit einer *Drittanbieteranwendung*, wie beispielsweise Moodle, benötigt.

6.1.3 Bewertung der Kernziele

In diesem Abschnitt werden die Kernziele dieser Arbeit bewertet. Die Grundlage für die qualitative Evaluation ist die in der Abbildung 29 [S.78] angeführte openLCMSI des SLHw-Projekts. Zu den *Clientanwendungen* gehören die *LTI Tools*, die *Editoren*, die *Lernbegleiter-App* und die *Recommendation Engine*. Die *Drittanbieteranwendungen* sind das LCMS-Repository, *Moodle* und *Learning Locker*. *Middleware Consumer* in diesem Szenario sind die *Beuth Hochschule für Technik Berlin* und *Fraunhofer FOKUS*.

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.

Das Repository liefert Dienstleistungen zum Erstellen und Verwalten von Kursen sowie Lerninhalten (LCMS). Die *Moodle*-Systeme werden zur Nutzerverwaltung (LMS) und Kommunikation zwischen Nutzern (COM) verwendet. Die *Drittanbieteranwendung Learning Locker* speichert Interaktionen von Nutzern während einer Lernsession (LRS).

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

Die Dynamisierung von Verbindungswegen wird durch das Routing ermöglicht. In der Abbildung 29 [S.78] werden die unterschiedlichen *Routes* der LMS, LRS und COM MWAs von den *LTI Tools*, den *Editoren*, der *Lernbegleiter-App* und der *Recommendation Engine* verwendet.

Bei den *LTI Tools* handelt es sich um *Rich Client Applications*, die sowohl von einem LMS als auch von der Lerngleiter-App verwendet werden können. Ein besonders wichtiges *LTI Tool* ist *openPresentation*, welches die Darstellung von Lernobjekten aus dem Repository ermöglicht.

Die Editoren erlauben neben dem *Hub*-Management auch das Erstellen und Verwalten von Lerninhalten und Kursen sowie Einstellungen zu den *LTI Tools*, wie zum Beispiel ein *Corporate Design*.

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.

Dies ist ein Problem, das in der aktuellen Version der MI nicht ausreichend gelöst wurde. In Zukunft wird geplant, dass ein *Middleware Consumer* mehrere *Presets* (Sammlungen von *Routes*) für *Virtuelle Organisationen* abrufen kann. Eine *Virtuelle Organisation* (VO) kann eine Institution oder ein Zusammenschluss von mehreren Institutionen sein.

Datenhoheit des Nutzers

In der Abbildung 29 [S.78] besteht keine direkte Kommunikation zwischen *Drittanbieteranwendungen*. Eine Verknüpfung der Daten erfolgt erst in der Präsentationsschicht beziehungsweise bei den *Clientanwendungen*.

Ein Nutzer authentifiziert sich gegenüber einer *Drittanbieteranwendung* clientseitig. Zu diesem Zweck wurde die *Bypass*-Authentifizierung vorgesehen. Die Verarbeitung der Zugangsdaten erfolgt nicht über die MI. Die Daten der gewünschten Institution werden nach seiner Zustimmung an die von ihm genutzte *Clientanwendung* übermittelt. Weitere Institutionen erhalten diese Daten nicht.

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*.

Ein *Middleware Consumer* identifiziert sich durch einen Token. In der aktuellen Systemumgebung ist der Token in dem Lernbegleiter hinterlegt. Dieses Szenario ist langfristig zu optimieren.

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.

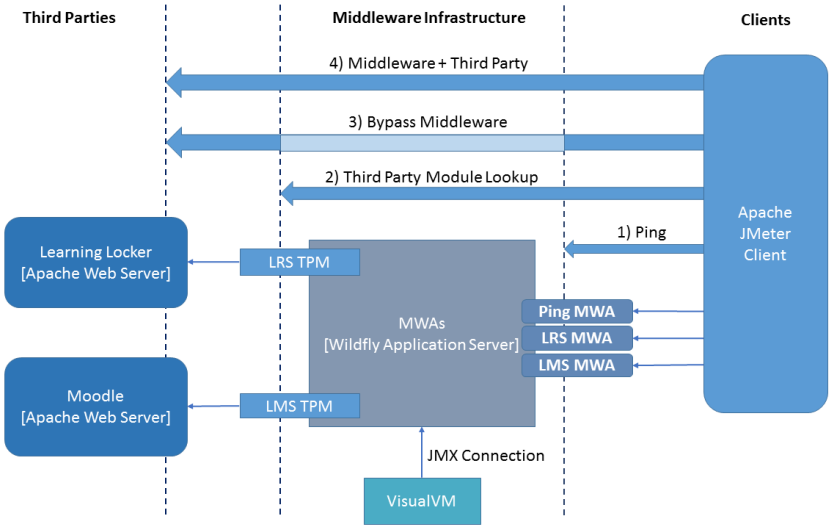

Abbildung 30: Test Environment - Middleware API (Eigene Darstellung)

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)

Hardware	Specification
CPU	I7-3770 @3,4 GHz (8 virtual cores)
RAM	16 GB
Harddrive	SSD 250 GB
Network Adapter	Intel(R) 82579LM

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.

Sowohl die *Apache*-Webserver als auch eine *MySQL*-Datenbank werden durch *XAMPP* bereitgestellt. In dem *XAMPP*-Setup wurden, wie auch bei *Wildfly*, die Standardkonfigurationen verwendet. Zusätzlich wird neben der *MySQL*-Datenbank eine *MongoDB* zur Verfügung gestellt.

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

Die Abbildung 30 [S.81] stellt vier Testszenarien dar. Grundsätzlich ist zu ermitteln, welchen Einfluss die MI auf die Verarbeitungszeit einer Anfrage von einer *Clientanwendung* bis zu einer *Drittanbieteranwendung* hat. In den Testszenarien werden stufenweise alle Architekturschichten durchlaufen und im Einzelnen betrachtet.

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.

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)

Testscenarios	Elapsed[ms]	Execution[ms]	Time[s]	CPU[%]	RAM[mb]
Ping (Test 1)	469,71	0,052	54,72	4	180
TP Lookup (Test 2)	485,99	0,510	55,94	4	180

Es wird ersichtlich, dass die Verarbeitungszeit der MWA (Execution) im Vergleich zu den Latzenzzeiten 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 Latenzzeiten 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)

Testscenarios	Elapsed[ms]	Execution[ms]	Time[s]	CPU[%]	RAM[mb]
Ping (Test 1)	666,39	0,052	84,39	5	180
TP Lookup (Test 2)	682,74	0,510	86,76	9	180

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.

Tabelle 12: Scenarios 3 and 4 with 250 Requests - 50 Threads (Eigene Tabelle)

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

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.

Bei der Betrachtung von *Learning Locker* wurde eine durchschnittliche Dauer von 2,17 Sekunden ermittelt. Wird zusätzlich die entsprechende MWA (LRS) verwendet, beträgt die Dauer im Durchschnitt 2,24 Sekunden.

Die Tabelle 13 [S.86] verdeutlicht den Datendurchsatz (Performance) für *Moodle* und *Learning Locker* für die Testszenarien 3 und 4. Berücksichtigt wird der Datendurchsatz der Response, da es sich um Dienstleistungen handelt, bei denen Daten abgerufen werden.

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

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

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])

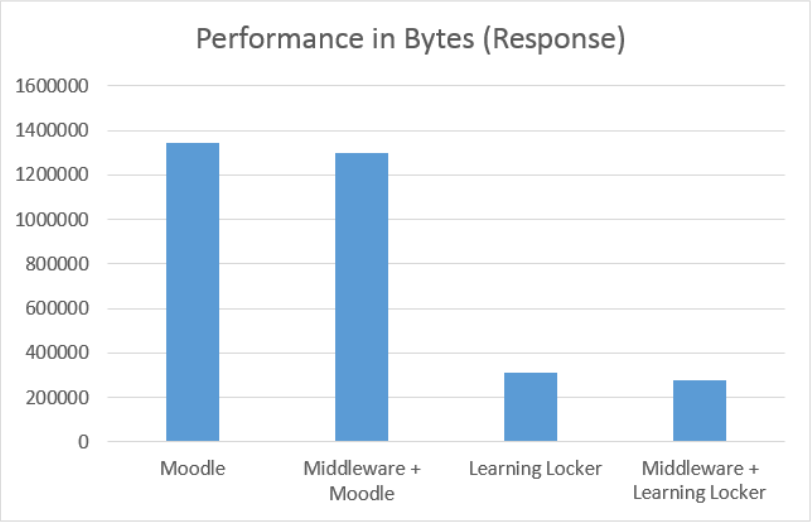

Abbildung 31: Performance - Middleware APIs vs. Third Parties (Eigene Darstellung)

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 excecution 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.

Die Performance der *Drittanbieteranwendung* hat einen starken Einfluss auf die gesamte Performance. Skalierbare Web-Technologien, welche beispielweise in MOOC-Architekturen Anwendung finden, sollten bevorzugt werden.

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

Tabelle 14: Performance factors (Eigene Tabelle)

No.	Performance factors
1	Third Party-Application performance and API compatibility
2	Network infrastructure and server setup
3	Datamodel (DTO / Entity)
4	Third Party Module (adapter)
5	Third Party Module Lookup

6.3 Zusammenfassung

In der Evaluation wurden qualitative und quantitative Softwarekriterien ausgewertet.

Die qualitativen Softwarekriterien zeigten den vielfältigen Einsatz der *Middleware Infrastructure* (MI). Es wurden im Rahmen des SLHw-Projekts drei Kurse in der *Handwerkskammer Berlin*, der *Technischen Universität Berlin* und der *Beuth Hochschule für Technik Berlin* angeboten. Insgesamt wurden 206 Nutzer eingeschrieben. Die Kernziele dieser Arbeit wurden weitestgehend erfüllt.

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.

Die quantitativen Softwarekriterien wurden durch eine Performance-Messung der MI evaluiert. Die Auswertung ist als Richtwert zu verstehen, da im Rahmen dieser Arbeit für Stresstests keine Produktionsumgebung gewährleistet werden konnte. Die Daten werden zudem durch die Latenzzeiten verzerrt.

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.

Auf diese Weise versuchte der Autor dieser Arbeit, den heterogenen Markt von E-Learning-Angeboten für Entwickler zu vereinheitlichen. Der aktuelle Prototyp dieser Arbeit ermöglicht es, Lernanwendungen zu entwickeln, die feingranular eine Vielzahl von Dienstleistungen aus E-Learning-Angeboten integrieren. Zudem wird das lebenslange Lernen gefördert, indem eine Lernanwendung Daten aus verschiedenen Institutionen, vorzugsweise Bildungseinrichtungen, anfordern kann. Sowohl der Zugriff auf die E-Learning-Angebote als auch die Wahl der Institution ist bei einer Lernanwendung dynamisch und erfordert nur einmalig einen höheren Implementierungsaufwand.

7.1 Fazit

Recherchen über die E-Learning-Branche haben aufgezeigt, dass viele verschiedene Arten von Drittanbieteranwendungen in der E-Learning-Branche angeboten werden. Die im Rahmen dieser Arbeit entwickelte MI kommuniziert direkt mit den Drittanbieteranwendungen, um deren Dienstleistungen zentral und interoperabel nutzen zu können. Aus diesem Grund war eine Betrachtung des E-Learning-Marktes relevant für eine Einschätzung, ob die Nutzung einer MI einen ausreichenden Mehrwert bietet.

Der Markt bei den monolithischen Systemen mit einer Eingrenzung auf *Learning Management Systeme* (LMS) ist stark heterogen. 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), *Ilias*, *Eduplone*, *SAKAI* und *WebCT* (Zedan and Al-Ajlan, 2005).

Neben den klassischen LMS-Angeboten werden vermehrt auch flexiblere serviceorientierte Lernangebote entwickelt. Zu diesen gehören *Personal Learning Environments* (PLE), bei denen ein Nutzer die Lernumgebung aktiv beeinflussen kann und im Zentrum der Systemumgebung steht sowie die *Massively Open Online Courses* (MOOC), welche eine sehr hohe Anzahl an Nutzern für einen Kurs anstreben. Auch der Markt bei den MOOCs ist heterogen. Es etablierten sich vier bekannte Anbieter von MOOC-Plattformen. Im Jahr 2015 wurde der Markt hinsichtlich der MOOCs durch die Anbieter *Coursera*, *Open edX*, *Udacity* und *FutureLearn* bestimmt (Peters et al., 2015).

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.

> In Clarke (2005) wurde die Content-Interoperabilität einer Vielzahl von Lernplattformen in Bezug auf den SCORM-Standard untersucht. Es zeigte sich, dass derzeit der Import und Export von Lernobjekten, die den SCORM-Standard erfüllen, zwischen den Lernplattformen nicht immer reibungslos funktioniert. (Frankfurth and Schellhase, 2006)

Die Qualität der Standards ist nicht die Kernproblematik. Vielmehr setzen Standards voraus, dass die Entwickler eines E-Learning-Angebots diese vollständig beziehungsweise ordnungsgemäß implementieren. Die Verantwortung wird aus Sicht des Autors an der falschen Stelle erwartet. Die entwickelte MI ermöglicht es, alle Drittanbieteranwendungen zu adaptieren und zentral Clientanwendungen einen Standard, wie zum Beispiel Kursstrukturen nach IMS CC, anzubieten.

Eine Motivation des Autors war es, langfristig eine branchenübergreifende MI zu entwickeln. Dies bedeutet, dass beliebig viele *Middleware APIs* (MWA) und entsprechende Standards zu verschiedenen Branchen außerhalb der E-Learning-Branche bereitgestellt werden. Dieses Vorhaben setzte eine starke Entkopplung der Architekturkomponenten voraus. Aus diesem Grund wurden die Recherchen auf serviceorientierte Architekturansätze ausgeweitet.

Zu Beginn dieser Recherche wurde das Potenzial von SOAs für die E-Learning-Branche ermittelt. Technisch werden individuelle Funktionalitäten ermöglicht, die typische Kernfunktionalitäten eines E-Learning-Angebots erweitern können. Organisatorisch kann durch eine Verknüpfung von Systemen eine Mehrarbeit bei interuniversitären Lehrkooperationen vermieden werden. Qualitativ ist durch eine Orchestration (Kopplung) von Services eine Verknüpfung zwischen verschiedenen Bereichen einer Bildungseinrichtung möglich, wie beispielsweise Bibliotheks- und Prüfungsverwaltungssysteme. Ökonomisch werden bereits bestehende Services genutzt, um spezielle E-Learning-Anwendungen entwickeln zu können. Der Aufwand für vollständige Eigenentwicklungen entfällt weitestgehend und nur die speziellen Funktionalitäten werden zusätzlich implementiert.

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 *Mircoservices* aufgegriffen.

Fowler and Lewis (2015) beschreiben *Microservices* als ein Konzept, bei dem Funktionalitäten bis zur Datenerhaltungsschicht von anderen Services getrennt werden. Dieses Vorgehen bildet somit den Gegenpart zu monolithischen Systemen.

Bei der Betrachtung verschiedener serviceorientierter Ansätze wurden viele Szenarien vorgestellt, die besonders in Hinsicht auf ein lebenslanges Lernen förderlich sind. Es wurden interuniversitäre Kommunikationen und Rahmenpläne nach dem Konzept von *Joint Degrees* beschrieben. Interessant war hierbei auch eine Verknüpfung zwischen verschiedenen LMS mit Kunden- und Mitarbeiterdatenbanken eines Arbeitgebers. Auf diese Weise könnte das Auffinden von personalisierten Lerninhalten für Mitarbeiter im Rahmen einer Fortbildung eine Erleichterung bieten.

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.

Die MI wurde in die openLCMSI des SLHw-Projekts integriert. Das Adaptieren von Drittanbieteranwendungen durch *Third Party Modules* ermöglichte eine interoperable Nutzung verschiedener E-Learning-Anwendungen. Durch das Routing wurden drei Kurse in der *Handwerkskammer Berlin*, der *Technischen Universität Berlin* und der *Beuth Hochschule für Technik Berlin* angeboten. Insgesamt wurden 206 Nutzer eingeschrieben.

Evaluiert wurden die qualitativen und quantitativen Softwarekriterien. Die qualitativen Softwarekriterien betrachteten die Kernziele aus der Einleitung, welche durch Recherchen begründet wurden. Bei den quantitativen Softwarekriterien handelte es sich um Performance-Messungen in einer kontrollierten Testumgebung. Die Evaluation lieferte zwei Ergebnisse, die als Richtwert dienen sollen.

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

Zum anderen wurde der Performanceeinfluss ermittelt, der bei dem Einsatz der MI entsteht. Zu diesem Zweck erfolgte ein Vergleich zwischen der direkten Nutzung einer Drittanbieteranwendung gegenüber einer Nutzung mit einer MWA. Im Falle von *Moodle* wurde eine Differenz im Datendurchsatz von 3% ermittelt. Bei der Integration von *Learning Locker* ist eine Differenz von 13% erfasst worden. Der Aufwand von 0,51 Millisekunden für das Routing und die in der Differenz vorhandenen Latenzzeiten sowohl durch den *Overhead* einer MWA als auch durch die allgemeine Netzwerkinfrastruktur führte zu der Annahme, dass die Codeperformance einer MWA im Vergleich zu externen Performanceeinflüssen zu vernachlässigen ist. Zu den externen Performanceeinflüssen gehörte neben der Netzwerkinfrastruktur die Performance der *Drittanbieteranwendung*, die Kompatibilität der Drittanbieterschnittstellen zu den MWA-Standards und das *Data Transfer Object* (DTO, Fowler et al., 2002, pp. 347-356).

7.2 Ausblick

Im Rahmen dieser Arbeit konnten aus zeitlichen und finanziellen Gründen nicht ausreichend E-Learning-Angebote über die MI bereitgestellt werden. Bei einer Weiterentwicklung sollten vorzugsweise E-Learning-Angebote mit einer hohen Marktabdeckung auf eine Integrierbarkeit überprüft werden. Hierbei ist vorerst zu recherchieren, ob externe Schnittstellen, wie beispielsweise Webservices, verfügbar sind. Auch der Zugriff auf die Ressourcen sowie die Berechtigungshierarchien sollten für interuniversitäre Anwendungsfälle geeignet sein.

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.

Neben den technischen Anforderungen sollten Fragen bezüglich des Datenschutzes und der Rolle als Dienstleistungsvermittler evaluiert werden. Zudem ist die Akzeptanz von Drittanbietern hinsichtlich *Middleware*-Architekturen zu ermitteln.

Die Authentifizierung von Nutzern erfolgt zwischen Client- und Drittanbieteranwendungen. Eine Route liefert Informationen zu einem Script mit geeigneter Logik für die Authentifizierung. Auf diese Weise werden keine Nutzerdaten in der MI übertragen.

Eine interuniversitäre Nutzung der Routen erfordert neue Anforderungen an die Authentifizierung. Eine Möglichkeit ist die Integration von *Single-Sign-On*-Mechanismen (SSO). *Lightweight Directory Access Protocol* (LDAP) sollte unverändert über die bisherige MI eingesetzt werden können. Komplexere Lösungen, wie beispielsweise OAuth, sind zu implementieren.

In dieser Arbeit wurde speziell die E-Learning-Branche betrachtet. Die MI ist ein abstraktes Konstrukt und in erster Linie für das Adaptieren und Routing verantwortlich. Auf Ebene der MWA besteht eine klare Trennung von Funktionalitäten durch *Microservices*. Um das Konzept der branchenspezifischen MWAs zu vertiefen, sollte eine weitere Branche integriert werden. Die E-Health-Branche erscheint vielversprechend, da beispielsweise der Markt bei Fitness-Apps sowie bei Anbietern von Fitnesstrackern (engl. *wearables*) sehr vielfältig ist (siehe Abbildung 32 [S.93]). Eine Erfassung von Vitaldaten ist zudem nicht nur auf sportliche Tätigkeiten begrenzt.

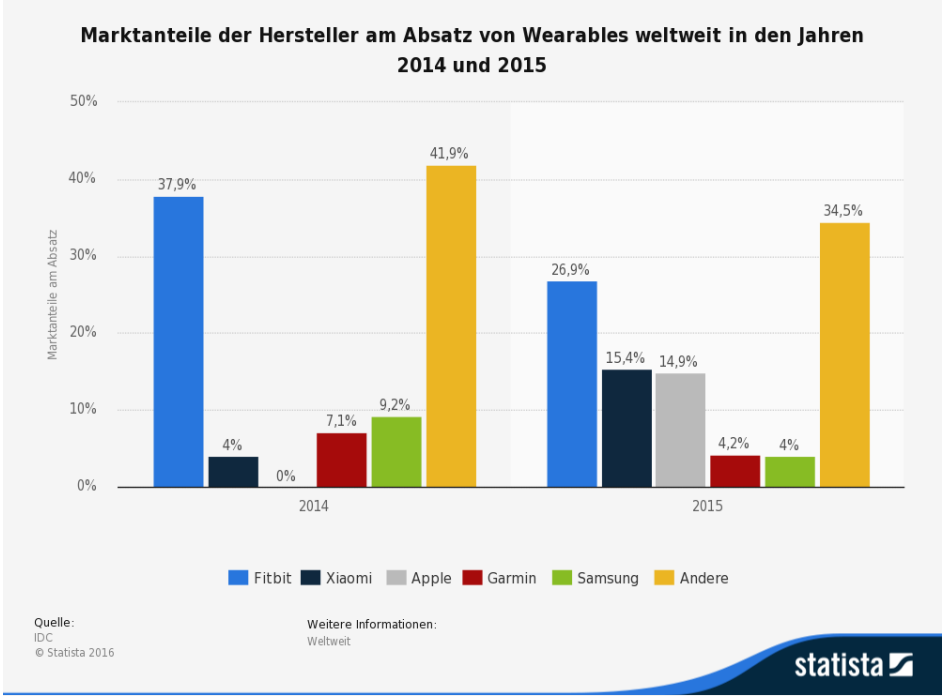

Abbildung 32: Marktanteile Hersteller Wearables 2014/2015 (Statista, 2015)

Eine Suche nach dem Begriff *Fitness* über den *Playstore* liefert derzeit eine Auswahl von 245 Fitness-Apps.[2] Für die Entwickler der Fitness-Apps wäre es von Vorteil, durch die Integration der MI dem Endnutzer eine größere Freiheit bei der Gerätewahl zu gewähren.

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

[2]Playstore Suchanfrage Stand Februar 2017

ren im Gesundheitswesen (Bundesregierung, 2015). In der *Telemedizin* sind vor allem Smartphones und Tablets dazu geeignet, als digitale Hilfsmittel flächendeckend telemedizinische Dienstleistungen bieten zu können. Vielversprechend erscheint dies unter anderem aufgrund langer Wartezeiten für medizinische Behandlungen und der ärztlichen Unterversorgung in peripheren Regionen. Besonders bei chronischen Wunden sind die Rückfallraten sehr hoch und die Krankheitsverläufe sehr lang, was eine kontinuierliche Beobachtung und Behandlung erfordert. Es gibt zunehmend Anbieter telemedizinischer Apps und Patienteninformationsseiten. Ein nachvollziehbares rechtliches Hindernis ist das Fernbehandlungsverbot, bei dem ärztliche Dienstleistungen nicht ausschließlich über indirekte mediale Wege erfolgen dürfen. (vgl. Augustin et al., 2016)

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 Patienteninformationsseiten 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.

Aus Sicht der Entwickler von Clientanwendungen besteht keine direkte Schnittstellenabhängigkeit zu Drittanbieteranwendungen. Die Verantwortung der Standardisierung wird zentralisiert und obliegt somit nicht den Firmeninteressen der Drittanbieter.

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

Aguirre, S. , Quemada, J. , & Salvachúa, J. . Work in progress - Developing Joint Degrees through e-Learning systems. *Proceedings - Frontiers in Education Conference, FIE*, pp. 19–20, 2008. ISSN 15394565. doi: 10.1109/FIE.2008.4720474.

Alur, D. , Malks, D. , & John, C. . *Core J2EE Patterns: Best Practices and Design Strategies*. 2001. ISBN 0130648841.

An, T.-S. , Dubois, F. , Manthey, E. , & Merceron, A. . Digitale Infrastruktur und Learning Analytics im Co-Design Stand der Forschung und Technik zu Learning Analytics. *Proceedings of DeLFI Workshops 2016 co -located with 14th e-Learning Conference of the German Computer Society (DeLFI 2016)*, pp. 8–17, 2016. ISSN 16130073.

Augustin, M. , Storck, M. , Lawall, H. , Heyer, K. , Protz, K. , chronischer Wunden Kristina Heyer, E. , Glaeske, G. , Diener, H. , & Katharina Herberger, P. . Chronische Wunden Mehr Qualität und Effizienz in der Versorgung chronischer Wunden Was gehört zu den Standards? *Gesellschaftspolitische Kommentare (GPK)*, 2:16, 2016.

Baumgartner, P. . Didaktik und Reusable Learning Objects (RLOs). *Campus 2004 - Kommen die digitalen Medien an den Hochschulen in die Jahre*, pp. 311–327, 2004.

Buchem, I. , Merceron, A. , Kreutel, J. , Haesner, M. , & Steinert, A. . Designing for User Engagement in Wearable-technology Enhanced Learning for Healthy Ageing. In: *Workshop Proceedings of the 11Th International Conference on Intelligent Environments*, pp. 314–324, 2015. ISBN 978-1-61499-530-2; 978-1-61499-529-6. doi: 10.3233/978-1-61499-530-2-314.

Bundesregierung. Entwurf eines Gesetzes für sichere digitale Kommunikation und Anwendungen im Gesundheitswesen. *Gesetzentwurf der Bundesregierung*, 2015.

Clarke, E. A. . Reuse and Repurposing of Resources for Content Exchange, including Technical Considerations on Interoperability. 2005.

Coates, H. , James, R. , & Baldwin, G. . A critical examination of the effects of learning management systems on university teaching and learning. *Tertiary Education and Management*, 2005. ISSN 13583883. doi: 10.1007/s11233-004-3567-9.

Dagger, D. , O'Connor, A. , Lawless, S. , Walsh, E. , & Wade, V. P. . Service-Oriented e-learning platforms: From monolithic systems to flexible services. *IEEE Internet Computing*, pp. 28–35, 2007. ISSN 10897801. doi: 10.1109/MIC.2007.70.

EdX. Open edX Architecture, 2016. URL `https://edx.readthedocs.io/projects/edx-developer-guide/en/latest/architecture.html`. Accessed 2017-02-13.

ELF. The E-Learning-Framework. URL `http://www.elframework.org/index.html`. Accessed 2017-02-17.

Fielding, R. T. . Representational State Transfer (REST). In: *Architectural Styles and the Design of Network-based Software Architectures*, chapter 5. 2000.

Foster, I. & Kesselman, C. . *The Grid 2: Blueprint for a New Computing Infrastructure*. Elsevier Ltd., 2 edition, 2003. ISBN 978-1558609334.

Fowler, M. . Inversion of Control Containers and the Dependency Injection pattern, 2004. URL https://www.martinfowler.com/articles/injection.html#InversionOfControl. Accessed 2017-02-13.

Fowler, M. & Lewis, J. . Microservices: Nur ein weiteres Konzept in der Softwarearchitektur oder mehr? *Objektspektrum*, pp. 1–7, 2015.

Fowler, M. , Rice, D. , & Foemmel, M. . *Patterns of Enterprise Application Architecture*. Edison Wesley, 2002. ISBN 0321127420.

Frankfurth, A. & Schellhase, J. . Potentiale serviceorientierter Architekturen für E-Learning-Infrastrukturen an Hochschulen. In: *DeLFI 2006: Die 4. e- Learning Fachtagung Informatik der Gesellschaft für Informatik e.V.(GI) Lecture Notes in Informatics (LNI) - Proceedings*, pp. 351–362, 2006. ISBN 978-3-88579-181-2.

Gamma, E. , Helm, R. , Johnson, R. , & Vlissides, J. . *Entwurfsmuster - Elemente wiederverwendbarer objekteorientierter Software*. Addison Wesley, 2011. ISBN 978-3-8273-3043-7.

Harmelen, M. , van. Personal Learning Environments. In: *Proceedings of the Sixth International Conference on Advanced Learning Technologies (ICALT'06)*, 2006. ISBN 0-7695-2632-2. doi: 10.1080/10494820701772645. URL http://www.cblt.soton.ac.uk/multimedia/PDFs/personallearningenvironments.pdf.

IETF. The OAuth 2.0 Authorization Framework: Bearer Token Usage. *The Internet Engineering Task Force (IETF)*, pp. 1–18, 2012.

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

IMS Global Learning Consortium. IMS Abstract Framework: White Paper, 2003. URL https://www.imsglobal.org/af/afv1p0/imsafwhitepaperv1p0.html. Accessed 2017-02-12.

IMS Global Learning Consortium. IMS Global Thin Common Cartridge® Profile: Implementation Guide (for CC v1.2 and v1.3), 2015a. URL https://www.imsglobal.org/cc/CCv1p0thin/ims_thinCC_impl-v1p0.html. Accessed 2017-02-13.

IMS Global Learning Consortium. IMS Question and Test Interoperability: Implementation Guide Version 2.2, 2015b. URL https://www.imsglobal.org/question/qtiv2p2/imsqti_v2p2_impl.html. Accessed 2017-02-13.

Ivanović, M. & Jain, L. C. . *E-Learning Paradigms and Applications: Agent-based Approach*. Springer, 2014. ISBN 978-3-642-41964-5. doi: 10.1007/978-3-642-41965-2.

JBoss. Chapter 20. The Virtual File System, 2008. URL https://docs.jboss.org/jbossmc/docs/2.0.x/userGuide/ch20.html. Accessed 2017-02-13.

Keen, M. , Acharya, A. , Bishop, S. , Hopkins, A. , Milinski, S. , Nott, C. , Robinson, R. , Adams, J. , & Verschueren, P. . Enterprise Service Bus and SOA patterns. In: *Patterns: Implementing an SOA Using an Enterprise Service Bus*, chapter 4, pp. 77–78. 2004. ISBN 0738490008.

Krauss, C. , Merceron, A. , & An, T.-S. . You might have forgotten this learning content! How the Smart Learning Project recommends appropriate Learning Objects. *International Journal on Advances in Intelligent Systems, vol 9*, 2016a.

Krauss, C. , Merceron, A. , An, T.-S. , Zwicklbauer, M. , & Arbanowski, S. . The Smart Learning Approach A mobile Learning Companion Application. *eLmL 2016 : The Eighth International Conference on Mobile, Hybrid, and On-line Learning*, pp. 13–16, 2016b.

Liu, X. L. X. , Saddik, a. E. , & Georganas, N. . An implementable architecture of an e-learning system. *CCECE 2003 - Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology (Cat. No.03CH37436)*, pp. 1–4, 2003. ISSN 0840-7789. doi: 10.1109/CCECE.2003.1225995.

Oracle. Lesson: Overview of JNDI. URL `http://docs.oracle.com/javase/tutorial/jndi/overview/index.html`. Accessed 2017-02-12.

Oracle. Introduction to Contexts and Dependency Injection for the Java EE Platform, 2013a. URL `http://docs.oracle.com/javaee/6/tutorial/doc/giwhb.html`. Accessed 2017-02-12.

Oracle. JAX-RS 2.0 API Specification, 2013b. URL `https://jax-rs-spec.java.net/nonav/2.0/apidocs/`. Accessed 2017-02-13.

Oracle. Enterprise Beans, 2013c. URL `http://docs.oracle.com/javaee/6/tutorial/doc/gijsz.html`. Accessed 2017-02-13.

Peters, G. , Sacker, D. , & Seruga, J. . A Comparative Analysis of MOOC - Australia ' s Position in the International Education Market 2 Foundations of Massive Open Online Courses. *Australasian Conference on Information Systems*, pp. 1–10, 2015.

Ritchie, C. . Inject external properties using CDI, Java and WildFly, 2015. URL `http://blog.chris-ritchie.com/2015/03/inject-external-properties-cdi-java-wildfly.html`. Accessed 2017-02-13.

Siemens, G. . Connectivism: A learning theory of the digital age. *International Journal of Instructional Technology and Distance Learning*, 2005.

Stal, M. . Using Architectural Patterns and Blueprints for Service-Oriented Architecture. *IEEE Software*, pp. 61, 2015. ISSN 1098-6596. doi: 10.1017/CBO9781107415324.004.

Statista. Share of learning management system (LMS) market worldwide in 2013, by vendor, 2013. URL `https://www.statista.com/statistics/501086/worldwide-learning-management-software-vendor-market-share/`. Accessed 17.02.2017.

Statista. Absatz von wearables weltweit nach Hersteller, 2015. URL `https://de.statista.com/statistik/daten/studie/515716/umfrage/absatz-von-wearables-weltweit-nach-hersteller/`. Accessed 17.02.2017.

TinCan. What is the Tin Can API? URL `https://tincanapi.com/overview/`. Accessed 2017-02-13.

Zedan, H. & Al-Ajlan, A. . E-Learning (MOODLE) Based on Service Oriented Architecture. In: *Proc. of the EADTU's 20th Anniversary Conference*, 2005. URL `http://www.cse.dmu.ac.uk/STRL/research/publications/pubs/2007/2007-23.pdf`.

Zhu, H. . Challenges to reusable services. In: *Proceedings - 2005 IEEE International Conference onServices Computing, SCC 2005*, pp. 243–244, 2005. ISBN 0769524087. doi: 10.1109/SCC.2005.36.

Anhang

A LRS Statement


```
"data":[
  {
    "id":"63b38664-8b32-4464-9be3-22beff5790aa",
    "timestamp":"2017-02-13T14:48:23.598800+00:00",
    "stored":"2017-02-13T14:48:23.598800+00:00",
    "version":"1.0.0",
    "verb":{
      "id":"https://w3id.org/xapi/adl/verbs/abandoned"
    },
    "actor":{
      "objectType":"Agent",
      "name":"teacher_@example.org",
      "mbox":"mailto:teacher_@example.org"
    },
    "object":{
      "objectType":"Activity",
      "id":"https://vfh143.beuth-hochschule.de/demo/openStudio/middleware/repository/modules/RklUXzAwMQ",
      "definition":{
        "type":"http://adlnet.gov/expapi/activities/module",
        "name":{
          "de-DE":"Fitnessübung+1"
        }
      }
    },
    "context":{ },
    "authority":{
      "objectType":"Agent",
      "name":"",
      "mbox":"demo@example.org"
    }
  }
```


Abbildung 33: LRS Statement (Eigene Darstellung)

B LMS Course


```
"data":[ ⊟
  { ⊟
    "metadata":{ ⊟
      "general":{ ⊟
        "identifier":[ ⊟
          { ⊟
            "catalog":"catalog",
            "entry":"20"
          }
        ]
      }
    },
    "organizations":[ ⊟
      { ⊟
        "identifier":null,
        "structure":"rooted-hierarchy",
        "item":{ ⊟
          "identifier":"root",
          "items":[ ⊟
            { ⊞ },
            { ⊞ }
          ]
        }
      }
    ],
    "resources":{ ⊟
      "I_897_R":{ ⊟
        "identifierref":null,
        "type":"lti",
        "title":"Grafische Benutzeroberfläche in JavaFX - Grundlagen",
        "description":null,
        "launchURL":"https://vfh143.beuth-hochschule.de/beuth/vfh/openPresentation/modules/SkZYX0dVSQ",
        "secureLaunchURL":null,
        "detailsURL":null,
        "icon":null,
        "secureIcon":null,
        "vendor":null
      },
      "I_896_R":{ ⊞ },
      "I_903_R":{ ⊞ },
      "I_902_R":{ ⊞ },
      "I_899_R":{ ⊞ },
      "I_898_R":{ ⊞ }
    }
  }
],
```


Abbildung 34: LMS Course (Eigene Darstellung)

C Hub Management

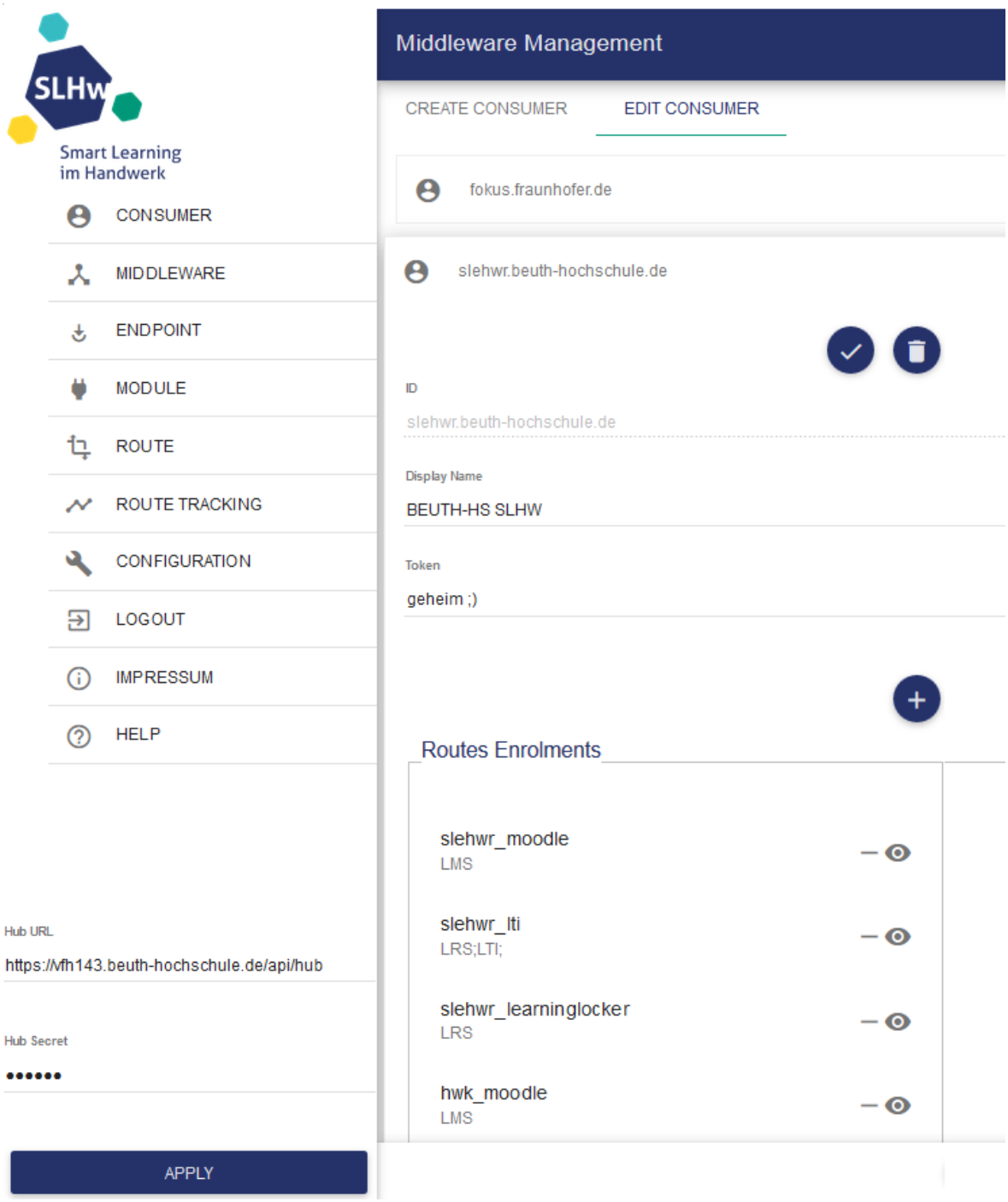

Abbildung 35: Edit Middleware Consumer (Eigene Darstellung)

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.

$Unterschrift:$ $Ort, Datum:$