Goethe-Universität Frankfurt

Diplomarbeit

Integration der Experience API in Autorensysteme für Web Based Trainings

eingereicht bei

Prof. Dr.-Ing. D. Krömker

Professur für Graphische Datenverarbeitung

von

Patrick Sacher

Eingereicht am: 09. September 2013

Eidesstattliche Versicherung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Literatur und Hilfsmittel angefertigt habe. Wörtlich übernommene Sätze oder Satzteile sind als Zitat belegt, andere Anlehnungen hinsichtlich Aussage und Umfang unter Quellenangabe kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist nicht veröffentlicht.

Frankfurt am Main, 09.09.2013

Ort, Datum	**Patrick Sacher**

Danksagung

Mein besonderer Dank gebührt Prof. Dr.-Ing. Detlef Krömker von der Goethe-Universität Frankfurt für die Betreuung und die Möglichkeit dieser Arbeit sowie meinem Betreuer Dipl.-Inf. David Weiß, der mir stets mit Rat und Tat zur Seite stand.

Ebenso danke ich meiner Familie und meinen Freunden für ihre Unterstützung während dieser Arbeit und meines gesamten Studiums.

Inhaltsverzeichnis

1 Einleitung

eLearning wird, wie alle anderen Lernformen auch, immer wieder von aktuellen Trends in der Technik und der Gesellschaft beeinflusst. Die aktuellen Trends zur Nutzung von mobilen Geräten und die schon längst in der Gesellschaft angekommene Nutzung von sozialen Medien im Internet, haben somit auch einen großen Einfluss auf die Anforderungen, die an zeitgemäße eLearning-Inhalte und Konzepte gestellt werden.

Besonders für *Autorensysteme für Web Based Trainings* stellt die Forderung, diese aktuellen Anforderungen des eLearnings zu erfüllen, eine zentrale Aufgabe dar. Autorensysteme für WBTs sollten immer aktuelle und geeignete Technologien in die mit ihnen erstellten WBTs integrieren, sodass der WBT-Autor in der Lage ist, mit den erstellten WBTs zeitgemäße Konzepte des eLearnings umzusetzen.

Wie in dieser Arbeit gezeigt wird, ist der bislang am weitesten verbreitete eLearning-de-facto-Standard *SCORM* nicht mehr in der Lage, diese aktuellen Anforderungen zu erfüllen. So müssen Autorensysteme für WBTs neuere Modelle und Technologien integrieren, um den aktuellen Anforderungen des eLearnings gerecht zu werden.

Die *Training and Learning Architecture*, die von der *Advanced Distributed Learning Initiative* entworfen wird, soll diese aktuellen Anforderungen des eLearnings erfüllen und ein Modell sowie Technologien für die Umsetzung zeitgemäßer eLearning-Inhalte und Konzepte bereitstellen. Die *Experience API* stellt dabei die Kernkomponente und technische Basis der Training and Learning Architecture dar.

Die Experience API ist aktuell die einzige Komponente der TLA für die bislang Spezifikationen in einer vorläufig vollständigen Version erschienen sind. Da es sich bei der Experience API um die technische Basis der TLA handelt, soll deren Konzeption und technische Umsetzung genau analysiert werden, um zu klären, ob die Experience API als technisches Fundament der TLA geeignet ist. Denn nur wenn alle von der TLA geforderten Anforderungen an die Experience API technisch sowie konzeptuell erfüllt werden, kann die TLA die Anforderungen eines zeitgemäßen Modells für aktuelle eLearning-Inhalte darstellen.

Ziel dieser Arbeit ist es herauszuarbeiten, welche Anforderungen aktuelle Trends an Konzepte und Techniken eines zeitgemäßen eLearnings stellen sowie zu untersuchen, ob diese mit Hilfe der TLA und speziell mit der Experience API zu realisieren sind. Anschließend soll eine prototypische Integration der Experience API in ein, von einem Autorensystem erstellten, WBT dazu dienen, die Analysen auch praktisch zu überprüfen und die notwendigen Schritte zur Implementierung der Experience API in ein Autorensystem für Web Based Trainings zu beschreiben.

2 Motivation

Der aktuell verbreitetste eLearning-de-facto-Standard SCORM basiert auf Anforderungen für eLearning-Inhalte aus den Jahren um 2000. Durch die rasante Entwicklung des Internets und allgemeiner IT-Technologien sind die mit SCORM realisierbaren Konzepte nicht mehr zeitgemäß.

So sind mit SCORM zum Beispiel mobile, native Apps sowie außerhalb eines *Learning Management Systems* durchgeführte Aktivitäten nicht erfassbar. Doch gerade soziale Plattformen sowie mobile Geräte werden in der Gesellschaft immer verbreiteter und sollten auch in Lernprozesse integrierbar sein können.

Die *Training and Learning Architecture* (inklusive ihrer technischen Basis, der *Experience API*) scheint ein vielversprechendes, neues Modell für eLearning-Inhalte bereitzustellen. Ob die TLA sowie die zur Umsetzung gewählten Techniken allerdings wirklich alle Anforderungen erfüllen können, soll untersucht werden.

Außerdem soll gezeigt werden, wie eine Integration der Experience API in ein mit einem Autorensystem erzeugtes WBT durchgeführt werden kann und mit welchem Aufwand sie verbunden ist.

Somit soll diese Arbeit einen Beitrag dazu leisten, die TLA und besonders die Experience API auf ihre Tauglichkeit zur Umsetzung eines modernen WBTs zu untersuchen.

Diese Frage ist einerseits für alle Entwickler von Autorensystemen für WBTs sowie allgemein für WBT-Autoren von Bedeutung. Des Weiteren ist die Experience API als Community-Driven-Project entstanden und auch in diesem Zusammenhang sind Kritiken oder Mängel, die gegebenenfalls im Verlauf dieser Arbeit herausgearbeitet werden, hilfreich, um die Entwicklung der TLA erfolgreich fortführen zu können.

3 Grundlagen

Zunächst werden Grundlagen, die zum Verständnis dieser Arbeit benötigt werden, behandelt.

Dabei wird auf zentrale Begriffe wie *eLearning* und *Web Based Training* eingegangen sowie der für diese Arbeit relevante eLearning-Standard *SCORM* erklärt.

3.1 eLearning

Definition des Begriffes *eLearning* aus [Eh11]:

> *„Der Begriff [eLearning] umfasst alle Formen des Lernens mit Hilfe elektronischer Medien. Sowohl online als auch offline. Es ist eine Lernform, bei dem die neuen Informations- und Kommunikationsmedien (Computer und Internet) in Lernarrangements eingebunden werden, entweder zur Unterstützung des Lernprozesses (als sogenannte „hybride“ Lernarrangements) oder als ausschließliche Form der Vermittlung.“*

eLearning kann demzufolge als zusätzliche oder ausschließliche Form der Wissensvermittlung genutzt werden.

Eine Besonderheit gegenüber anderen Lernformen ist die enge Verknüpfung mit modernen Technologien der Informatik. Die didaktischen Möglichkeiten der Lehrenden werden durch den Stand der Technik begrenzt. Im Umkehrschluss fließen moderne Technologien und Trends der Informatik in die didaktischen Konzepte des eLearnings ein und können dadurch neue Vermittlungsmethoden möglich machen.

Spricht man also von eLearning, so muss man die didaktischen Konzepte sowie die technologischen Mittel als eine Einheit ansehen, die sich gegenseitig bedingen, ergänzen und zusammen den Begriff des eLearnings definieren.

In dieser Arbeit liegt der Schwerpunkt auf der technischen Komponente des eLearnings und den daraus resultierenden Möglichkeiten zur Erstellung neuer didaktischer Konzepte für das *Computerunterstütze Lernen*.

Die Schreibweise des Begriffes *eLearning* unterscheidet sich in den verschiedenen Publikationen. Es gibt unter anderem die Variationen *E-Learning*, *ELearning* oder *eLearning*. In dieser Arbeit wird letztere Schreibweise verwendet.

3.2 Web Based Training

Ein *Web Based Training*, abgekürzt WBT oder übersetzt *Webbasiertes Lernen,* ist eine weiterentwickelte Form des *Computer Based Training* (*CBT*).

Ein CBT bezeichnet ein Lernprogramm, ein Lernpaket oder einen Kurs, der von einem Lernenden auf einem Computer (oder einem anderen elektronischen Gerät) ausgeführt wird, um sich einen Lerninhalt, meist im Selbststudium, zu erschließen.

Meist handelt es sich bei einem CBT um eine per CD-ROM oder DVD verteilte, native Applikation, die auf dem Computer des Lernenden ausgeführt wird.

Die Besonderheit des WBTs, im Gegensatz zu einem CBT, ist die Verteilung und die Ausführung über das Internet oder ein anderes Netzwerk (Intranet) und die dadurch theoretisch mögliche systemunabhängige Ausführung über einen Web-Browser.

Durch Verwendung webbasierter Technologien kann ein WBT auch Funktionen zur Vernetzung und Zusammenarbeit von Lernenden bereitstellen sowie den Lehrenden Möglichkeiten bieten, um den Ablauf des Lernprozesses zu begleiten und nachzuvollziehen.

Die Umsetzung eines WBTs geschieht mit Hilfe von HTML(5), JavaScript, Flash, Java oder anderer gängiger Web-Technologien, die in einem Web-Browser ausgeführt werden können. Auch interaktive Elemente wie Fragen oder Videos können dadurch realisiert werden.

3.3 Autorensysteme für Web Based Trainings

Autorensysteme für WBTs sind Entwicklungswerkzeuge, mit deren Hilfe Lehrende digitale Lernangebote/Kurse erstellen können. Die Lehrenden werden in diesem Bezug auch als *Autoren* bezeichnet.

Autorensysteme sind häufig Template-basiert, sodass Autoren auch ohne größeres technisches Vorwissen mit ihnen WBTs erstellen können. Es existieren aber durchaus auch Autorensysteme, die dem Autor weniger Hilfestellung bieten. In diesem Fall hat der Autor zwar größere Freiheiten, muss sich allerdings im Gegenzug meist mehr mit der Technik des WBTs beschäftigen.

Oft unterstützen Autorensysteme gängige *eLearning-Standards* wie *SCORM*, *AICC* oder das *IMS Content Packaging*, wodurch die erstellten WBTs von den Autoren besser verteilt und ausgetauscht werden können (siehe Kapitel 0).

Beispiele für Autorensysteme für WBTs sind: Adobe Captivate [1], eXe Learning[2] oder das LernBar Autorensystem[3].

[1] http://www.adobe.com/de/products/captivate.html - (kommerziell)
[2] http://www.exelearning.de/ - (Open-Source)
[3] http://www.studiumdigitale.uni-frankfurt.de/et/LernBar/ - (kommerziell/kostenlos für Universitäten und Schulen)

3.4 Learning Management Systeme

Learning Management Systeme (LMS) sind spezielle web-basierte Softwarepakete, die Lehrenden Möglichkeiten bieten, um:

1. Lerninhalte zu strukturieren
   (Erstellung von Kursräumen, Gliederung von Inhalten, …)
2. Teilnehmer / Lernende zu verwalten.
   (Einteilung in Lerngruppen, Zuordnung in spezielle Kurse, …)
3. Inhalte und Aktivitäts-Module bereitzustellen
   (Bilder, Videos, WBTs sowie Wikis, Foren oder Chats)
4. Lernerfolge aufzuzeichnen und auszuwerten

Im Gegenzug erhalten Lernende:

1. Einfachen Zugriff auf die bereitgestellten Inhalte
2. Orts-/ und Zeit-unabhängigen Zugang zu den Lerninhalten
3. Interaktionsmöglichkeiten mit anderen Teilnehmern oder den Kursleitern

Die Features, die verschiedene LMS enthalten, variieren stark. So gibt es sehr komplexe und mächtige LMS sowie auch LMS, die sich auf ein Minimum von Funktionen, wie zum Beispiel nur das Bereitstellen von Kurs-Materialien und eine Rechte-Verwaltung, beschränken.

Meist unterstützen LMS gängige *eLearning-Standards*, um WBTs zu importieren oder um die Lernerfolge aufzeichnen zu können (mehr dazu in Kapitel 0).

Beispiele für LMS sind: Moodle[4], Olat[5] oder ILIAS[6].

3.5 eLearning-Standards

Durch die Tatsache, dass ein WBT allein durch die Ausführbarkeit in einem Web-Browser definiert ist, kommt es zu dem Problem, dass jedes WBT eine andere Struktur und andere Techniken verwenden kann. Dies gibt den Autoren zwar einen großen Gestaltungsspielraum beim Entwickeln von WBTs, jedoch entsteht dadurch das Problem, dass eine Wiederverwendung in unterschiedlichen LMS oder eine automatische Katalogisierung und Beschreibung des WBTs nicht möglich ist.

Aus diesem Grund wurden eLearning-Standards entwickelt, die die Wiederverwendbarkeit von WBTs in verschiedenen LMS oder allgemein die Austauschbarkeit ermöglichen sollen.

[4] http://moodle.de/ - (OpenSource)
[5] http://www.olat.org/ - (OpenSource)
[6] http://www.ilias.de/ - (OpenSource)

Abbildung 1 - Aufgabe von eLearning Standards

Die *Advanced Distributed Learning Initiative (ADL)* definiert die Ziele einer Standardisierung von WBTs und LMS, durch bestimmte Eigenschaften (Abilities), die ein WBT haben sollte, um kostengünstig und nachhaltig produziert werden zu können:

Accessibility: *the ability to locate and access instructional components from one remote location and deliver them to many other locations.*

Interoperability: *the ability to take instructional components developed in one location with one set of tools or platform and use them in another location with a different set of tools or platform*.

Durability: *the ability to withstand technology changes without redesign, reconfiguration or recoding*.

Reusability: *the flexibility to incorporate instructional components in multiple applications and contexts*.[7]

Die verbreitetsten eLearning-Standards (meist De-facto-Standards[8]), die die Ziele der *Erreichbarkeit*, *Interoperabilität*, *Beständigkeit* und *Wiederverwendbarkeit* umzusetzen versuchen, sind:

- **IMS Content Packaging & Learning Object Metadata Standard** [IM07]
  vom *IMS Global Learning Consortium*
- **CMI - Computer Managed Instructions - Guidelines for Interoperability** [AI93]
  vom *AICC - Aviation Industry CBT Committee*
- **LOM - Standard for learning object metadata** [IE02]
  von der *Learning Object Metadata (LOM) working group*
- **SCORM – Sharable Content Object Reference Model** in Version 1.2 [Ad01] und 2004 [Ad09] von der *Advanced Distributed Learning Initiative (ADL)*

Es existiert eine Vielzahl von anderen eLearning-Standards, deren Vergleich und Aufzählung für diese Arbeit jedoch nicht direkt relevant ist. Stattdessen wird sich diese Arbeit auf SCORM und die direkt damit verbundenen eLearning-Standards konzentrieren.

[7] [Ad01](Overview) - Seite 29 – Kapitel 1.5.1 High-Level Requirements

[8] De-facto Standards (oder Industriestandards) sind Standards, die sich auch ohne die Zertifizierung durch ein Normungsverfahren bei ihrem Praxis-Einsatz als nützlich und sinnvoll erwiesen haben.

Dies hat einerseits den Hintergrund, dass die *Experience API*, die das Hauptthema dieser Arbeit bildet, als direkter Nachfolger von SCORM entwickelt worden ist und andererseits SCORM der eLearning-Standard mit der höchsten Popularität ist.

So konnte Corinne Montandons in ihren empirischen Untersuchungen zum Bekanntheitsgrad von verschiedenen eLearning-Standards von 2004 und 2006 zeigen, dass SCORM den höchsten Bekanntheitsgrad und die höchste Verbreitung unter eLearning-Autoren hat (siehe [Co06] und [Co04]).

Als aktuellere Quelle für die Verbreitung und Popularität von SCORM kann man die Anzahl der LMS und Autorensysteme die eine SCORM-Unterstützung anbieten heranziehen. So kann man zum Beispiel auf http://www.elearningatlas.com eine Übersicht über viele verfügbare eLearning Angebote wie LMS und eLearning-Tools finden.

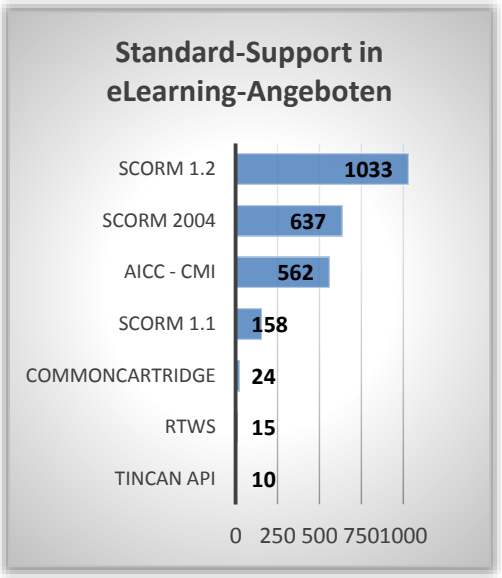


Abbildung 2 – Anzahl der Unterstützungen von eLearning-Standards der auf http://www.elearningatlas.com gelisteten eLearning Angebote.

Betrachtet man hierbei den Filter zur Suche nach unterstützten eLearning-Standards, so zeigen die Zahlen deutlich, dass SCORM in den verschiedenen Versionen eindeutig in den meisten eLearning-Produkten integriert wurde. Der AICC-CMI-Standard hat auch eine sehr hohe Verbreitung in eLearning-Produkten, allerdings wurde der AICC-Standard (wie gleich gezeigt wird) in SCORM integriert, wodurch diese hohe Verbreitung nicht direkt als Konkurrenz gesehen werden kann.

Der hohe Bekanntheitsgrad von SCORM ist primär darin begründet, dass SCORM genaugenommen kein eigenständiger Standard ist, sondern ein Referenz-Modell für eLearning-Inhalte, der vorhandene, andere Standards kombiniert und teilweise erweitert. Nichtsdestotrotz wird SCORM durch seine hohe Popularität als De-facto Standard angesehen.

Auf Grundlage der Arbeit von Corinne Montandon hat Jan Gellweiler in seinem Vortrag [Ja05] eine Übersicht über das Zusammenspiel und die Kombination der verschiedenen eLearning-Standards erstellt. Hier sieht man deutlich, dass SCORM aus der Kombination von vielen eLearning-Standards entstanden ist, aber auch andere eLearning-Standards Ideen und Technologien untereinander austauschen.

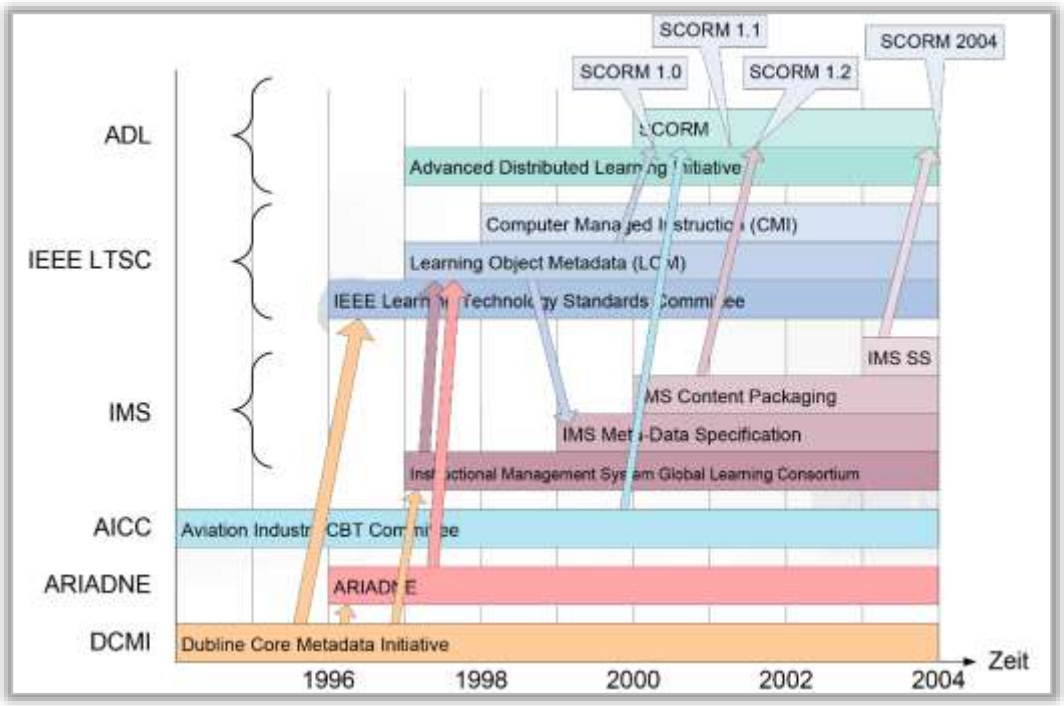


Abbildung 3 - Verbindungen zwischen den populärsten eLearning-Standards

Diese Verbindungen zwischen den einzelnen Standards kommen nicht zufällig zustande. So gibt es seit Langem mehrere Arbeitsgemeinschaften zwischen den einzelnen Konsortien ADL, IMS, ACC, IEEE, …, in denen Ergebnisse ausgetauscht und zusammen weiterentwickelt werden.

3.5.1 SCORM – Sharable Content Object Reference Model

Das *Sharable Content Object Reference Model* (im weiteren Verlauf der Arbeit nur noch *SCORM* genannt) wird von der *Advanced Distributed Learning Initiative* (ADL) entwickelt.

Die ADL wurde 1997 durch das US-Verteidigungsministerium gegründet. Aufgabe der ADL war es, ein Referenz-Modell für eLearning-Inhalte zu erstellen, um diese besser verwalten und wiederverwenden zu können. Hintergrund der Gründung waren erhoffte Einsparungen an Kosten, die durch ständige Neuentwicklungen und Anpassungen der zur Ausbildung genutzten eLearning-Inhalte an neue LMS anfielen.

Wie bereits erwähnt, verzichtete die ADL bei der Entwicklung von SCORM auf eine komplette Neuentwicklung aller Komponenten, sondern kombinierte und erweiterte stattdessen schon vorhandene eLearning-Standards so, dass möglichst alle vorgegebenen Ziele erfüllt wurden.

3.5.1.1 SCORM 1.2

SCORM 1.2 war die erste öffentliche Version von SCORM und erschien im Oktober 2001. Die Spezifikation besteht aus drei Büchern, die die verschiedenen Komponenten des Modells beschreiben[9]:

1. **Overview**
2. **Content Aggregation Model** - *CAM*
3. **Run-Time Environment** – *RTE*

Das erste Buch *Overview* beinhaltet lediglich eine kurze Einführung in SCORM und seine Komponenten, sowie Erklärungen über die Ziele von SCORM.

Das zweite Buch *Content Aggregation Model* definiert Regeln, wie die einzelnen Dateien der Inhalte strukturiert werden sollten und wie man mit Hilfe von Metadaten die Inhalte beschreiben kann. Die daraus resultierenden SCORM-Pakete können dann in alle SCORM-kompatiblen LMS importiert werden.

Das dritte Buch *Run-Time Environment* beschreibt, wie die Kommunikation zwischen LMS und WBTs durch eine JavaScript-API stattfinden kann und beschreibt das Daten-Modell der zu übermittelnden Daten.

3.5.1.2 SCORM 2004

Im Januar 2004 erschien eine neue Version von SCORM unter dem Namen SCORM 2004.

Die Spezifikation zu SCORM 2004 besteht aus 4 Büchern[9]:

1. **Overview**
2. **Content Aggregation Model** - *CAM*
3. **Run-Time Environment** – *RTE*
4. **Sequencing and Navigation** – S&Q

Die Bücher 1-3 sind im Grunde nur Updates zu den Büchern aus SCORM 1.2. Große Änderungen am Inhalt finden sich nur in Buch 3 – Run-Time Environment, indem das Daten-Modell einem Update unterzogen wurde und einige Datenfelder andere Größen sowie Namen zugewiesen bekommen haben.

Das neue, vierte Buch enthält neue Spezifikationen zur Art, wie zwischen den einzelnen Bestandteilen des SCORM-Paketes navigiert werden kann. Dabei handelt es sich um Erweiterungen zu Buch 2, dem Content Aggregation Model. Zusätzlich zur Struktur und den Metadaten der Bestandteile des SCORM-Paketes, können zusätzlich Regeln angegeben werden, welche Inhalte anderen folgen oder ob bestimmte Voraussetzungen, wie ein erfolgreicher Abschluss, erfüllt sein müssen, um zum nächsten Lernobjekt navigieren zu können.

Es folgten drei weitere Updates der Spezifikation, wobei meist nur im Bereich von *Sequencing and Navigation* kleinere Änderungen vorgenommen wurden.

[9] SCORM-Spezifikationen als ZIP-Pakete verfügbar: SCORM 1.2: [Ad01] und SCORM 2004: [Ad09]

Aktuell ist *SCORM 2004 – 4th Edition*, veröffentlicht 2009, die letzte erschienene Version von SCORM 2004.

3.5.2 Die Bestandteile von SCORM

Da, wie in Kapitel 3.5.1.2 erwähnt, die Unterschiede zwischen den Hauptfunktionen von SCORM 2004 und SCORM 1.2 nur im Detail zu finden sind, wird in dieser Arbeit nicht immer explizit zwischen den beiden Versionen unterschieden. Sollte also in folgenden Abschnitten die Rede von Funktionen von SCORM sein, so sind damit Funktionen gemeint, die in SCORM 1.2 sowie SCORM 2004 identisch sind oder nur minimal voneinander abweichen.

Zum Verständnis dieser Arbeit ist nur das Grundgerüst von SCORM wichtig und dieses ist in beiden Versionen gleich, sodass eine Unterscheidung der verwendeten Version nicht immer notwendig ist.

3.5.2.1 SCORM - Content Aggregation Model

Abbildung 4 - Content Packaging

Das Content Aggregation Model beschreibt, wie die Dateien eines Lernpakets (WBTs) organisiert werden sollen, sodass ein reibungsloser Austausch und eine hohe Wiederverwendbarkeit des WBTs erreicht wird.

Hierfür wird der *IMS Content Packaging & Learning Object Metadata Standard* in Version 1.1.2 vom *IMS Global Learning Consortium* verwendet. [IM07]

Dieser besteht aus einigen einfachen Regeln und Richtlinien, die hier kurz umrissen werden.

Alle Dateien, die zum WBT gehören, werden in einer Datei gebündelt. Dazu verwendet der Standard ein einfaches ZIP-Format. Ein SCORM-Paket ist somit ein ZIP-Archiv, das alle benötigten Dateien enthält.

Als weitere Regel gilt, dass keine Dateien nachgeladen werden dürfen. Alle benötigten Dateien müssen also in dem SCORM-Paket vorhanden sein. Ansonsten wäre das Ziel der Austauschbarkeit des WBTs gefährdet.

Zusätzlich zu den vom WBT benötigten Dateien enthält das SCORM-Paket eine XML-Datei mit dem Namen *„imsmanifest.xml“* inklusive seiner *XML Schema Definitions Dateien (*.xsd)*.

In dieser *imsmanifest.xml-Datei* werden die einzelnen Elemente des SCORM-Paketes beschrieben und definiert.

```
<?xml version="1.0" encoding="UTF-8"?>
   <manifest
      identifier="com.scorm.manifesttemplates.scorm12"
      version="1"
      xmlns="http://www.imsproject.org/xsd/imscp_rootv1p1p2"
      xmlns:adlcp="http://www.adlnet.org/xsd/adlcp_rootv1p2"
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xsi:schemaLocation="
         http://www.imsproject.org/xsd/imscp_rootv1p1p2 imscpi_rootv1p1p2.xsd
         http://www.imsglobal.org/xsd/imsmd_rootv1p2p1 imsmd_rootv1p2p1.xsd
         http://www.adlnet.org/xsd/adlcp_rootv1p2 adlcp_rootv1p2.xsd">
   <metadata>
      <schema>ADL SCORM</schema>
      <schemaversion>1.2</schemaversion>
   </metadata>
   <organizations default="ORG-ID">
      <organization identifier="ORG-ID">
         <title>Organisation Title</title>
         <item identifier="ITEM-ID" identifierref="RES-ID" isvisible="true">
            <title>Item Title</title>
         </item>
      </organization>
   </organizations>
   <resources>
      <resource
         identifier="RES-ID" type="webcontent" adlcp:scormtype="sco"
         href="index.html" />
   </resources>
</manifest>
```

Code 1 - Beispiel für eine einfache imsmanifest.xml mit einem einzigen definierten SCO

Die kleinste Einheit eine SCORM-Pakets ist ein einzelnes *Sharable Content Object (SCO)* oder ein *Asset*, wenn es sich um nicht-interaktive Elemente (zum Beispiel Bilder) handelt. Alle SCOs sollten als *Ressourcen*, mit ihrer relativen Position im ZIP-Archiv, in der imsmanifest.xml, definiert werden. Anschließend können einzelne Ressourcen zu *Organizations* zusammengefasst werden sowie durch Metadaten beschrieben werden.

Durch diese Bündelung aller Dateien des WBTs in einem ZIP-Archiv und der Beschreibung der Struktur und der Metadaten des WBTs in der imsmanifest.xml, lassen sich einzelne SCORM-Pakete in LMS, die diesen Standard unterstützen, importieren, austauschen oder können per Email verschickt werden und auch offline ausgeführt werden. Ebenso können diese SCORM-Pakete jederzeit bearbeitet, ergänzt oder anders wiederverwendet werden.

SCORM Run-Time Environment

Die SCORM Run-Time Environment Spezifikation besteht aus zwei Teilen. Es wird eine ECMAScript-basierende API beschrieben, mit der ein WBT mit einem LMS kommunizieren kann und Daten untereinander ausgetauscht werden können. Weiterhin wird die Form der austauschbaren Daten festgelegt – das SCORM Data Model.

SCORM Run-Time Environment – API

In der *SCORM Run-Time Environment* Spezifikation wird eine *ECMAScript*[10]-basierende API beschrieben, die das LMS bereitstellen muss, und zu der sich das WBT verbinden muss.

Beim Start des WBTs muss also zum Beispiel eine JavaScript Funktion des WBTs nach der RTE-API des LMS suchen. Diese API muss sich nach der Spezifikation entweder auf der gleichen oder auf einer Parent-Webseite des WBTs in einem der übergeordneten DOM-Elemente befinden. Für das WBT bedeutet das, dass es entweder in einem Frame oder iFrame innerhalb des LMS gestartet werden kann oder in einem PopUp-Fenster, das vom LMS aufgerufen wurde.

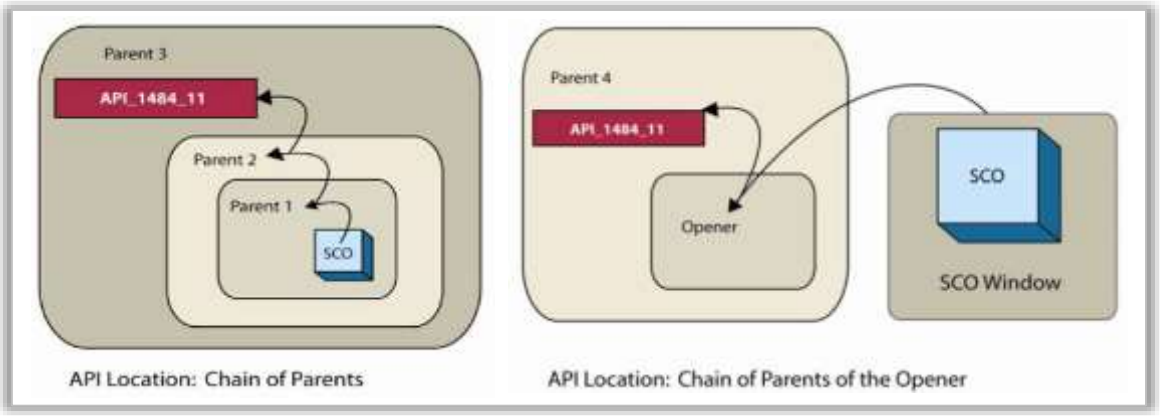


Abbildung 5 - Erlaubte Orte für die vom LMS bereitgestellte RTE-API und wie das WBT diese findet

Wurde die API gefunden, sollte der erste übertragene Befehl ein Aufruf von *initialize()* sein. Anschließend kann das WBT Daten an das LMS senden oder Daten aus früheren Sitzungen von dort anfordern.

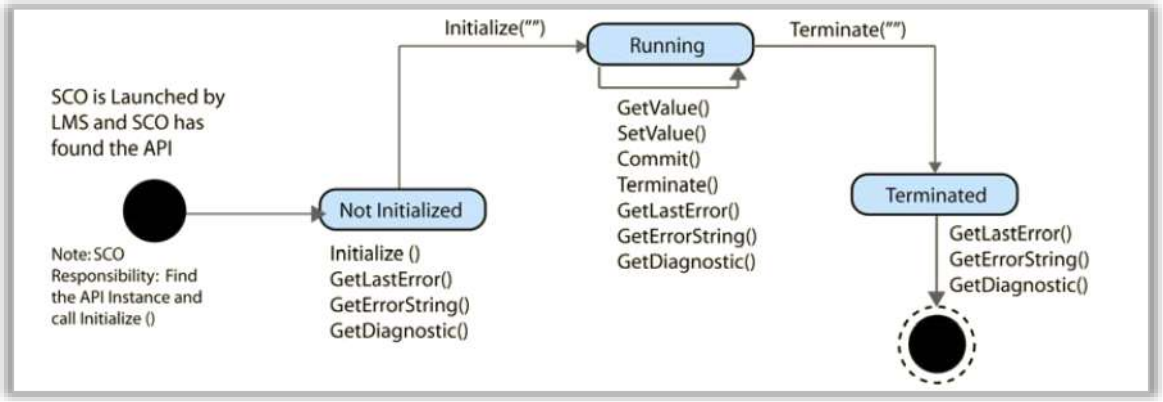


Abbildung 6 - Zustandsdiagramm einer RTE-API Verbindung

Die Spezifikation macht dabei keinerlei Angaben darüber, wie die API im LMS implementiert oder wie danach mit den Daten umgegangen werden soll. Es werden einzig feste Funktionen definiert, auf die das LMS nach bestimmten Regeln und Rückgabewerten reagieren muss.

[10] ECMAScript ist ein Standard für Script-Sprachen, der zum Beispiel in JavaScript oder ActionScript umgesetzt wird.

Ebenso müssen sich beide Seiten der Verbindung an die vom RTE-Data Model festgelegten Daten-Definitionen halten. Unbekannte oder nicht in der Spezifikation definierte Werte dürfen nicht übertragen werden.

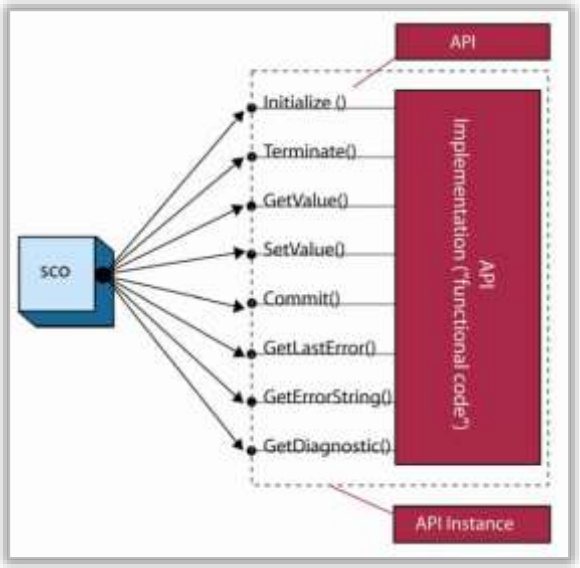


Abbildung 7 – Verfügbare Funktionen, die die RTE-API bereitstellen muss.

SCORM Run-Time Environment – Data Model

Das *RTE-Data Model* verwendet den Standard *CMI - Computer Managed Instructions Guidelines for Interoperability* der AICC zur Definition der Daten, die ein LMS und WBT untereinander austauschen können.

Die AICC veröffentlichte den CMI-Standard 1993. Ziel war es, Daten aus den von Piloten im Pilotentraining absolvierten eLearning-Einheiten zu erheben und diese von Auswertungsprogrammen einzulesen, um die Teilnehmer anschließend aufgrund ihrer Leistungen automatisch bewerten zu können.[11]

Dafür wurden feste Datenstrukturen geschaffen, die alle nötigen Informationen über eine eLearning-Einheit festhalten sollten. Auf [Ru13] sind alle Daten-Definition für die verschiedenen SCORM-Versionen aufgelistet und bequem einsehbar.

```
cmi.core.student_id      (CMIString (SPM: 255), RO)
   Identifies the student on behalf of whom the SCO was launched
cmi.core.student_name    (CMIString (SPM: 255), RO)
   Name provided for the student by the LMS
cmi.core.score.raw       (CMIDecimal, RW)
   Number that reflects the performance of the learner relative to the range
   bounded by the values of min and max
cmi.core.score.max       (CMIDecimal, RW)
   Maximum value in the range for the raw score
```

Code 2 - Beispiele für Definitionen des CMI-Data Models von SCORM 1.2

Alle Datenfelder haben das Präfix *cmi.* und sind sehr restriktiv definiert, was den Datentyp betrifft. So ist eindeutig festgelegt, welche Zeichen in einem String vorkommen dürfen und welche maximale Länge die Daten haben dürfen.

Zusätzlich zu den *core*-Daten des Kurses (*UserID*, *Score.Max*, usw…) ist es möglich Interaktionen (*Interactions*) wie zum Beispiel Fragen oder Lernziele (*Objectives*) zu definieren und zu protokollieren.

[11] Man könnte dabei von einer frühen Version von Learning Analytics (Kap. 4.1.2) sprechen.

3.5.2.2 SCORM – Zusammenfassung

Der SCORM-Standard ermöglicht es durch die Spezifikation des *Content Aggregation Models*, dass SCORM-kompatible WBTs von einem LMS importiert werden können. Nach dem Aufruf im LMS mittels der *Run-Time Environment* Spezifikation können sie mit diesem kommunizieren und Daten über den Kursablauf austauschen.

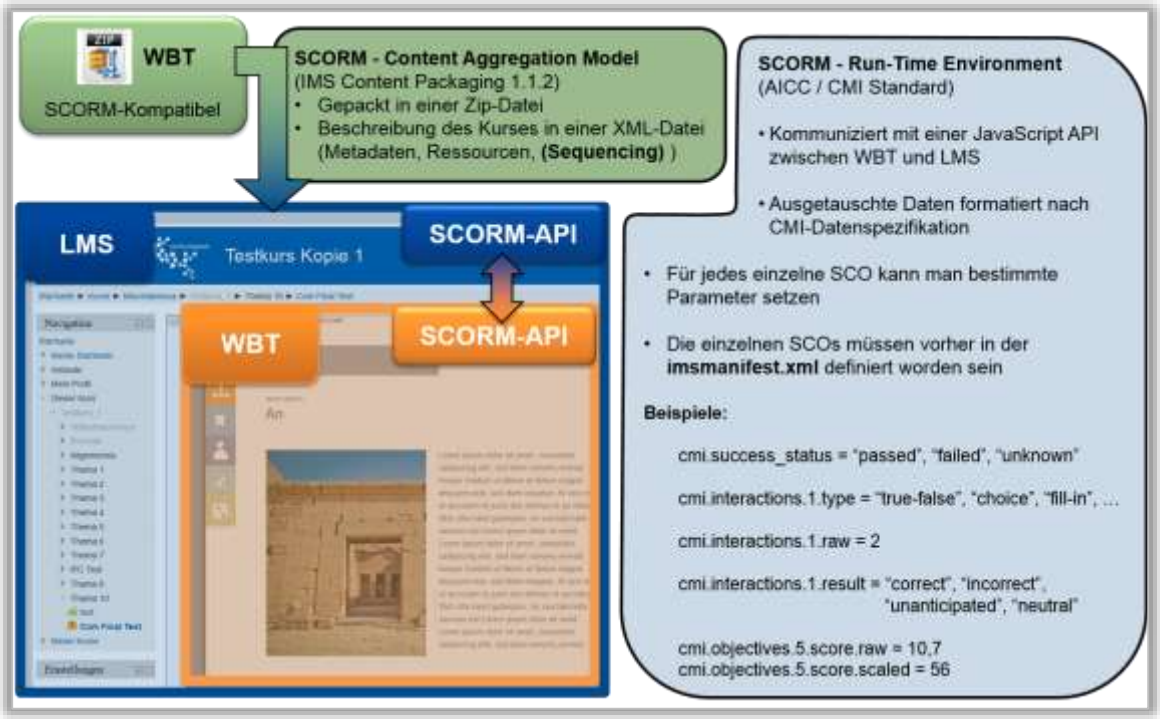


Abbildung 8 - SCORM Übersicht

4 eLearning – State of the Art

Nach der Einführung in alle zum Verständnis dieser Arbeit wichtigen Themen, sollen nun aktuelle Themen und Technologien des eLearnings vorgestellt werden.

Dabei wird auf aktuelle Trends im eLearning eingegangen, die gänzlich neue Anforderungen an die bestehenden Konzepte und Technologien stellen.

Diese Anforderungen werden anschließend auf WBTs übertragen und es wird überprüft, ob die Anforderungen mit dem bisher dominierenden eLearning-Standard *SCORM* noch realisierbar sind.

Abschließend wird die *Training and Learning Architecture* und besonders die *Experience API*, die Kernkomponente der TLA, vorgestellt. Die TLA befindet sich zum Zeitpunkt dieser Arbeit noch in der Entwicklung, verspricht aber Mittel zur Umsetzung der neuen Anforderungen an das eLearning bereitzustellen.

4.1 Aktuelle Trends im eLearning

Wie die Informatik und allgemein alle Lebensbereiche, so wird auch eLearning von aktuellen technologischen und gesellschaftlichen Trends beeinflusst. Diese haben Auswirkungen auf die Erwartungen der Lernenden und somit auch auf die Anforderungen an die Konzepte im eLearning.

Eine Einschätzung von aktuellen Trends, die das eLearning in den nächsten Jahren beeinflussen könnten, findet man zum Beispiel im jährlich erscheinenden *Horizon Report*. Dieser wählt jedes Jahr sechs Trends aus, stellt diese kurz vor und erläutert ihren möglichen Einfluss auf das eLearning. Weiterhin gibt er eine Zeitschätzung ab, wann aus den Trends allgemein gebräuchliche Methoden werden könnten.

Im *Horizon Report* 2012 [Th12] und 2013 [Th13] wurden folgende Top-Trends analysiert:

Zeithorizont	2012	2013
1 Jahr oder weniger	Mobile Apps	Massive Open Online Courses
1 Jahr oder weniger	Tablet Computing	Tablet Computing
2 bis 3 Jahre	Game-Based Learning	Games und Gamifizierung
2 bis 3 Jahre	Learning Analytics	Learning Analytics
4 bis 5 Jahre	Gesture-Based Computing	3D Printing
4 bis 5 Jahre	Internet of Things	Wearable Technology

Im Folgenden werden die wichtigsten Trends, die den Horizon Report schon seit einigen Jahren anführen und die für diese Arbeit von besonderer Relevanz sind, vorgestellt und daraus zeitgemäße Anforderungen an das eLearning entworfen.

Dabei werden *MOOCs* beispielhaft für die Vernetzung der Lernenden, *Learning Analytics* für den Bedarf an Möglichkeiten zur Auswertung von Lerner-Daten und die Themen *Mobile Apps* und *Tablet Computing*, zusammengefasst unter dem Begriff des *mobilen Lernens*, besprochen.

4.1.1 Massive Open Online Courses (MOOCs)

Unter *Massive Open Online Courses* (MOOCs) versteht man im Internet durchgeführte Lehrveranstaltungen, an denen eine große Anzahl von Personen teilnimmt[1]. Dabei unterscheidet man zwischen zwei unterschiedlichen Arten von MOOCs.

Extension MOOCs (xMOOCs) ergänzen oder ersetzen komplette Lehrveranstaltungen durch im Internet bereitgestellte Materialien, die die Teilnehmer dort bearbeiten können. Dabei können die Teilnehmer in speziellen LMS bereitgestellte Inhalte und Materialien jederzeit und ihrem eigenen Lerntempo entsprechend abrufen und bearbeiten. Die Lerninhalte werden dabei durch Texte und Videos vermittelt und anschließend durch Quizze oder Aufgaben, die einzureichen sind, bewertet. Zusätzlich werden meist Foren angeboten, in denen sich Teilnehmer untereinander und mit den Lehrenden austauschen können. Das MIT oder Stanford[2] bietet zum Beispiel die meisten seiner Lehrveranstaltungen, die regulär an der Universität angeboten werden, zusätzlich als MOOC für die Öffentlichkeit an. Somit hat jeder die Möglichkeit diese Kurse kostenlos zu bearbeiten und abzuschließen.

Connectivist MOOCS (cMOOCs) dagegen sind MOOCs, die ihren Fokus auf die Vernetzung der Teilnehmer und auf die Schaffung von Inhalten während der Laufzeit des MOOCs setzen. So werden meist nur ein Thema, eine geeignete Plattform und gegebenenfalls Auftaktbeiträge für einzelne Phasen des xMOOC vom Veranstalter vorgegeben. Die Inhalte sollen die Teilnehmer dann durch eigene Veröffentlichungen in Blogs, Tweets, Videoportalen oder Podcasts erstellen und durch Teilnahme an Video-Konferenzen, Chats oder Kommentaren diskutieren.[3]

cMOOCs nutzen die Mittel, die Web2.0- und Social-Media-Plattformen wie Blogger, Facebook oder Twitter bereitstellen, damit die Teilnehmer sich untereinander vernetzen können. Beiträge können weitergeleitet oder empfohlen werden, ebenso kann man einem bestimmten Teilnehmer folgen, wenn man seine Artikel sehr schätzt oder eben untereinander kommunizieren und über verschiedene Themen diskutieren.

Anstatt eines Abschlusszertifikates, wie bei den xMOOCs, können die Teilnehmer meist sogenannte *Badges* sammeln. Dies sind Auszeichnungen, die sie nach einer bestimmten Anzahl von getätigten Aktivitäten bekommen.

[1] Mehr Informationen zu MOOCs in [Br12]

[2] http://ocw.mit.edu/ und http://see.stanford.edu/

[3] Zum Beispiel:
http://opco12.de/ - OpenCourse zum Thema „Trends im E-Teaching – Der Horizon Report unter der Lupe“

Hauptunterschied und somit auch Hauptproblem der cMOOCs ist die von den cMOOCs geforderte Vernetzung von Teilnehmern und die Verteilung der Inhalte über die verschiedenen Plattformen im Internet. Hier besteht die Herausforderung darin, diese komplexen Vorgänge sinnvoll zu protokollieren und zu strukturieren. In xMOOCs und klassischen veranstaltungsbegleitenden LMS sind die Lehrinhalte sowie die Teilnehmer zentral in LMS angesiedelt. In cMOOCs werden die Inhalte auf den verschiedenen Web-Plattformen veröffentlicht und diskutiert. Andere cMOOC-Teilnehmer und die Veranstalter müssen informiert werden, wenn von einem Teilnehmer ein Beitrag zum cMOOC getätigt wird. Ebenso müssen die Ergebnisse im Verlauf zusammengetragen und ausgewertet werden.

➔ Die Möglichkeit, dezentral verteilte Aktivitäten der Lernenden/Teilnehmer zu protokollieren, stellt eine Anforderung dar, die moderne eLearning-Technologien und -Konzepte zu lösen in der Lage sein sollten.

4.1.2 Learning Analytics

Learning Analytics beschreibt die Analyse und Auswertung von Daten und Profilen, die von einem Lernenden während eines Lernprozesses anfallen.

Dabei kann man Learning Analytics mit den Methoden der Marktforschung vergleichen. In der Marktforschung werden enorme Mengen an Kundendaten (*Big Data*) erhoben und ausgewertet, um den Konsumenten zielgerichtete Kaufempfehlungen zukommen zu lassen, oder gezielt Produkte den Kundenwünschen anzupassen.

Auf Learning Analytics übertragen bedeutet dies, dass während der Bearbeitung eines Lerninhaltes so viele Daten wie möglich über die Art und Weise, wie der Lernende die gestellten Aufgaben bearbeitet, gesammelt werden sollen. Dabei sollen nicht nur Daten über die Aktivitäten im bereitgestellten Lerninhalt selbst gesammelt werden, sondern auch über den Lernenden und seine Aktivitäten außerhalb. Hat der Lernende auf einer anderen Internetseite Informationen zum bearbeiteten Thema eingeholt? Hat er sich mit anderen Personen über ein soziales Netzwerk vernetzt und ausgetauscht?

Die dabei anfallenden Daten sollen anschließend dafür verwendet werden, den Lerninhalt gegebenenfalls zu optimieren, aber auch, um ihn besser kategorisieren und einordnen zu können und ihn eventuell anderen Lernenden anbieten zu können, die die gleichen Eigenschaften wie der ursprüngliche Lernende hat.

Ebenso können die Daten dafür verwendet werden genauere Profile über die Lernenden zu erstellen. Diese Profile können von Lehrenden dafür genutzt werden, um Aktivitäten besser einschätzen und bewerten zu können, andererseits auch dafür, um dem Lernenden selbst ein besseres Bild seines Lernfortschrittes aufzuweisen. [4]

[4] Weitere Anwendungsbeispiele und Informationen zu Learning Analytics zum Beispiel in [ED11]

Dabei muss darauf geachtet werden, dass die Daten, die zum Beispiel während eines Lernprozesses in einem LMS gesammelt werden, nicht dort verbleiben, sondern auch nach außen verfügbar sind. So sollte dem Lernenden die Möglichkeit gegeben sein, seine erreichten Lernfortschritte in ein von ihm angelegtes Profil für *Lebenslanges Lernen* oder in ein anderes LMS übertragen zu können.

➔ Eine Anforderung an das aktuelle eLearning ist, dass Daten, die über einen Lernenden protokolliert werden, feiner und aussagekräftiger werden müssen sowie nach der Erhebung auch nach außen zugänglich und wiederverwendbar gemacht werden sollten.

4.1.3 mLearning und mobile WBTs

mLearning ist eine Unterart des eLearnings und beschreibt das mobile Lernen. Im Horizon Report können die Trends *Mobile Apps* und *Tablet Computing* unter diesem Begriff zusammengefasst werden.

Allgemein kann man unter mobilem Lernen das Lernen unterwegs mit einem Buch oder einem ausgedruckten Vorlesungs-Skript verstehen. Es beschreibt also jedes Lernen, das extern und nicht am traditionellen Vermittlungsort des Wissens (Schule, Universität) stattfindet und nicht zwingend mit Technik verbunden sein muss.

Im eLearning-Bereich wird mLearning als das Lernen mit mobilen Geräten wie Smartphones, Tablet-PCs oder eBook-Readern verstanden. In diesem Kontext spielt die Ortsunabhängigkeit des Lernens immer noch eine große Rolle, aber es werden auch Situationen berücksichtigt, in denen der Lernende direkt an traditionellen Orten der Wissensvermittlung mit mobilen Geräten arbeitet und diese in die Wissensvermittlung mit einbezogen werden.

Eine aktuelle Definition des mLearnings ist:

> *„mLearning behandelt jede Art des Lernens, das geschieht, während der Lernende sich nicht an einem festen, vordefinierten Ort befindet sowie Lernen, das durch Verwendung von Lerninhalten stattfindet die durch mobile Technologien bereitgestellt werden.“* [5]

Somit ist ein *mobiles Web Based Training* eine Variante des Web Based Trainings, das technisch sowie inhaltlich für die mobile Nutzung optimiert wurde.

Für ein mobiles WBT müssen daher einige andere Regeln als für ein normales WBT gelten:

1. Es dürfen für ein mobiles WBT nur Technologien verwendet werden, die auf dem Zielgerät auch unterstützt werden.

[5] Sinngemäß übersetzt aus [MO03]

2. Der Aufbau und das Design des mobilen WBTs muss an die meist kleineren und auf Touch-Bedienung ausgerichteten Geräte wie Smartphones oder Tablet-PCs angepasst werden.[6]
3. Da das Lernen in einer mobilen Umgebung Störungen unterworfen sein kann, die zu einer häufigen Unterbrechung des Lernens führen können, sollte bei der Entwicklung des Lerninhaltes darauf geachtet werden, dass dieser in „kleinen Häppchen“, sogenannten Nuggets, behandelt werden kann. Dies fordert eine spezielle Aufarbeitung des Lerninhaltes vom Autor.

Wie bereits gezeigt, wird das Thema „mobiles Lernen“ im Horizon Report seit einigen Jahren als einer der wichtigsten Trends eingeschätzt. Betrachtet man andere Statistiken, so zum Beispiel die App-Downloads für Smartphones der letzten Jahre (Abbildung 9), so kann man auch hier einen starken Anstieg erkennen, der die zunehmende Verbreitung und Nutzung von mobilen Endgeräten unterstreicht.

Abbildung 9 – Kumulierte Anzahl der weltweit heruntergeladenen Apps (in Mrd.)

Mobile Applikationen

Bei der Entwicklung eines mobilen WBTs steht vor der Entscheidung, welche Technologien verwendet werden können, eine allgemeine Analyse der mobilen Betriebssysteme, die von einer mobilen Applikation unterstützt werden sollten, da diese die verwendbaren Technologien primär beeinflussen.

Marktführend sind hierbei die Betriebssysteme iOS, das auf Apples iPhone- und iPad-Geräten zur Verfügung steht, sowie Googles Betriebssystem Android, das sich auf einer Vielzahl von Smartphones und Tablets der verschiedensten Hersteller befindet.

Andere bekannte Betriebssysteme sind Symbian, Blackberry OS und Windows Phone.

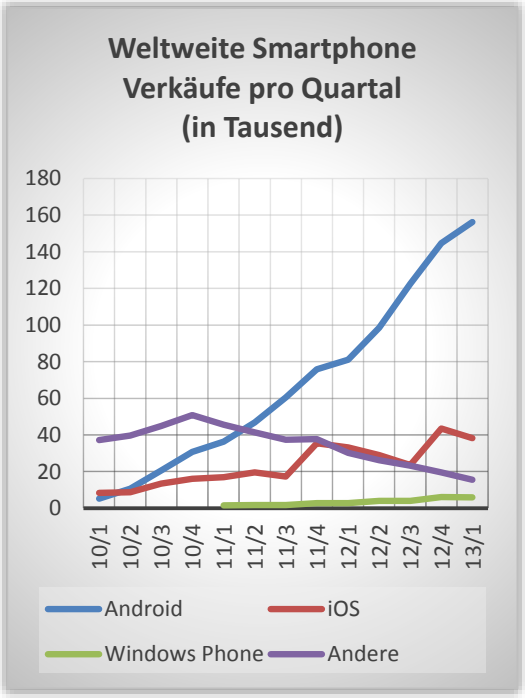


Abbildung 10 - Weltweiter Smartphone Verkauf pro Quartal (in Tausend)

[6] Weitere Informationen zu den speziellen Anforderungen an das Design von mobilen WBTs finden sich in [Be12].

Bis auf Geräten mit Windows Phone, das in Version 8 erst Ende 2012 erschienen ist, sinken die Verkaufszahlen der Geräte mit weniger verbreiteten Betriebssysteme.

Daraus kann man schließen, dass man bei der Entwicklung von mobilen Applikation besondere Rücksicht auf eine Kompatibilität mit Android-, iOS- und Windows Phone-basierenden Geräte achten sollte.

Im Allgemeinen kann man bei der Entwicklung von Applikationen für mobile Endgeräte (sogenannten Apps) zwischen drei Arten unterscheiden:

Native Apps

Native Apps sind direkt für das entsprechende Betriebssystem (OS) der Geräte geschriebene Applikationen, die sich installieren und ausführen lassen und Zugriff auf alle Gerätefunktionen erhalten können. Dabei wird zur Entwicklung das SDK der jeweiligen Plattform verwendet. Ein Austausch der Apps zwischen unterschiedlichen Betriebssystemen wie zum Beispiel iOS und Android ist hierbei nicht möglich.

Der Vertrieb von nativen Apps geschieht über die offiziellen oder alternativen *App-Stores* der jeweiligen Geräte.

WebApps

WebApps sind speziell an mobile Geräte angepasste Webseiten, die mit einem, auf dem mobilen Gerät installierten, Web-Browser aufgerufen werden. Dabei setzen WebApps meist auf *Responsive Design*, um das Layout der Webseite automatisch an verschiedene Displaygrößen und Geräte anpassen zu können. Um möglichst viele mobilspezifische Gerätefunktionen wie Touch-Steuerung, GPS und Kamera unterstützen zu können, bedient man sich hierbei moderner Web-Techniken wie HTML5, CSS3 und JavaScript. Die Kompatibilität zu den verschiedenen Browsern und mobilen Geräten kann man durch Verwendung von Bibliotheken wie *jQuery Mobile*[7] oder *Sencha Touch*[8] erhöhen.

Das Ziel einer WebApp ist es somit eine Applikation im Internet bereitzustellen, die, wie jede andere Webseite auch, mit geringem Entwicklungs-Aufwand auf jedem Smartphone-Modell ausgeführt werden kann. Eine WebApp kann somit nicht auf dem Gerät installiert werden, sondern muss von dem Nutzer über einen Link im Browser geöffnet werden. Da es sich im weitesten Sinne um eine erweiterte Internetseite handelt, muss eine Verbindung mit dem Internet auf dem Gerät vorhanden sein.

Hybride Apps

Hybride Apps versuchen eine Brücke zwischen WebApps und nativen Apps zu schlagen. Hybride Apps sind, wie Native Apps, gerätespezifische Apps, die jedoch nicht mit den

[7] http://jquerymobile.com/ - (Open Source, kostenfrei nutzbar)
[8] http://www.sencha.com/products/touch - (kommerziell)

SDKs der jeweiligen Betriebssysteme entwickelt wurden, sondern mit speziellen Frameworks wie PhoneGap[9] oder Titanium[10].

Dabei findet die Entwicklung wie bei WebApps mit HTML5, CSS3 und JavaScript statt. Zusätzlich hat man jedoch die Möglichkeit die vom Framework bereitgestellte API zu nutzen, mit der man auf spezielle Gerätefunktionen direkt zugreifen kann. Die so entwickelte App kann dann für mehrere OS gleichzeitig als native App veröffentlicht werden und die API-Befehle des Frameworks werden dabei automatisch in die jeweilige systemeigene Sprache übersetzt.

Hybride Apps können sich somit wie normale, native Apps für das Gerät verhalten und aussehen. Sie können direkt installiert und lokal ausgeführt werden, allerdings können sie weiterhin kleine Einschränkungen in der Nutzung von speziellen Gerätefunktionen haben.

Abbildung 11 - Unterschiede verschiedener App-Typen

Mobile WBTs

WBT-Autoren haben bei der Entwicklung eines mobilen WBT somit die Wahl, ob sie eine WebApp, eine hybride App oder eine native App erstellen wollen.

Mit der Entwicklung einer **WebApp** sind WBT-Autoren in der Lage, ein WBT nach traditioneller Definition (Ausführung im Browser) umzusetzen und durch Anpassungen am Design für mobile Geräte aufzubereiten. Die WebApp kann durch Verwendung spezieller Frameworks optisch und funktional so gestaltet werden, dass sie sich fast so verhält und aussieht wie eine gerätespezifische App. Allerdings muss die WebApp im Browser aufgerufen werden, was die Funktionalität auf die vom Browser und den aktuellen Web-Technologien unterstützten Möglichkeiten beschränkt.

[9] http://phonegap.com/ - (Open Source, kostenfrei nutzbar)
[10] http://www.appcelerator.com/platform/titanium-platform/ - (Open Source, kostenfrei nutzbar)

Entwickelt man dagegen eine **native App**, ist für den Lernenden der Zugriff auf den Lerninhalt schneller und flexibler zu erreichen. Es wird keine Internetverbindung vorausgesetzt, der Lerninhalt fügt sich nahtlos in das verwendete Betriebssystem ein und kann alle verfügbaren Funktionen und Komponenten des Gerätes nutzen.

Vieles davon können zwar auch **hybride Apps** leisten, jedoch mit Einschränkungen auf der technischen Seite. Handelt es sich um komplexere Applikationen, wie Simulationen oder auch Spiele mit Lernhintergrund (Game Based-Learning), so ist eine Ausführung im Browser als WebApp oder hybride App meist nicht möglich, da dieser nicht die erforderlichen Technologien bereitstellt.

Man kann somit zusammenfassen, dass aus Entwicklungsperspektive die WebApp oder die hybride App zu präferieren wären, da diese die geringsten Entwicklungskosten verursachen. Ausnahmen bilden komplexe Inhalte, die direkten Zugriff auf die Geräte-Hardware benötigen. In diesem Fall würde man eine native App präferieren.

Aus Anwendersicht steht jedoch die native App ganz klar an der Spitze. Sie bietet den höchsten Komfort für den Benutzer und bricht nicht mit dem Geräte-Design. Die Performance ist größer als bei WebApps oder hybriden Apps und alle Gerätefunktionen können zur Anreicherung des Lerninhaltes verwendet werden. Außerdem setzt eine native App keine Internetverbindung voraus und kann somit jederzeit und überall ausgeführt werden. Gerade bei mobilen Geräten ist dies von Vorteil, da die Wahrscheinlichkeit eines Verbindungsabbruchs, durch kurzweilig schlechten Empfang, sehr viel höher ist als bei nicht-mobilen WBTs.

WBTs können somit als hybride oder WebApp umgesetzt werden, solange der Autor sich den Einschränkungen bei der Entwicklung bewusst ist und die Vermittlung des Lerninhalts nicht darunter leidet. Gerade im Bereich der Wissensvermittlung sollte die Verwendung einer mobilen Applikation so einfach und zugänglich wie möglich sein und nicht durch Einschränkungen in der Bedienung oder der Technik beeinträchtigt werden. So sollte ein mobiles WBT, wenn möglich, als native App zur Verfügung gestellt werden, da diese dem Autor die meisten Möglichkeiten zur Entwicklung und dem Lernendem die beste Lernerfahrung bietet. [11]

➔ Eine Anforderung, die an das aktuelle eLearning gestellt werden muss, ist somit, dem Lernenden auf dem Gerät seiner Wahl einen orts- und zeitunabhängigen Zugang zu Lerninhalten zu ermöglichen.

[11] Weitere Informationen zu den Unterschieden sowie Vor- und Nachteile der verschiedenen App-Arten finden sich in [Be12], sowie in Philipp Maskes Büchern [Ma12a] & [Ma12b].

4.1.4 Anforderungen aktueller Trends an das eLearning

Aus den untersuchten aktuellen Trends im eLearning (mLearning, MOOCs und Learning Analytics) kann man drei konkrete Anforderungen an aktuelle eLearning-Konzepte und -Techniken definieren:

- ➔ Aktivitäten von Lernenden sollten auch außerhalb eines LMS protokollierbar und bewertbar sein, sodass auch externe Tools wie soziale Plattformen in den Lernprozess integriert werden können.
- ➔ Die erfassten Daten sollten aussagekräftiger werden, um auch LMS-externe Aktivitäten gut beschreiben zu können. Ebenso sollten die Daten nicht nur im LMS vorhanden sein, sondern nach außen erreichbar gemacht werden, um sie besser wiederverwerten zu können.
- ➔ Die Technik zur Protokollierung von Aktivitäten sollte Geräte- und auch Browserunabhängig verwendbar sein, sodass auch *Out-Of-Browser* Szenarien wie native Apps oder Serious Games möglich werden.

So ist das Hauptthema der untersuchten Trends die Dezentralisierung der Inhalte und Daten. Lange Zeit konzentrierte sich eLearning auf das Bereitstellen und Bearbeiten von Lerninhalten in LMS. Durch die zunehmende Vernetzung der Menschen im Internet und die Vielzahl von Quellen und Aktivitäten, die außerhalb eines LMS als produktive Ergänzung des Lernprozesses hinzugezogen werden können, wird die Forderung nach einer Möglichkeit auch diese Daten protokollieren zu können immer lauter. Würde dies erreicht, könnten auch Lehrende Inhalte außerhalb von LMS bereitstellen und somit auch andere Tools als die, die das LMS bereitstellt, in den Lernprozess einbinden. Inhalte sowie erhobene Daten sollten demzufolge auch außerhalb eines LMS verwendbar sein.

Sind die Lernaktivitäten durch Verwendung einer geeigneten technischen Basis dann erst einmal vom LMS losgelöst und ausgeführte Aktivitäten trotzdem protokollierbar, entstehen auch in der Entwicklung von WBTs völlig neue Möglichkeiten. Wird das LMS nicht mehr unbedingt zur Einbindung von Inhalten benötigt, müssten WBTs auch nicht mehr im Browser ausführbar sein. Native Apps, die den Lernenden auch das vollständig mobile Lernen möglich machen, wären dann ebenso realisierbar wie Serious Games oder Simulationen.

4.2 Die Training and Learning Architecture (TLA) - Ein neuer eLearning-Standard

Auch die ADL, die mit dem eLearning-Standard SCORM, den aktuell am häufigsten verwendeten eLearning-Standard entworfen hat, hat erkannt, dass viele der neuen Anforderungen an das eLearning mit SCORM nicht mehr realisiert werden können.

SCORM basiert auf Anforderungen für eLearning-Inhalte aus den Jahren um 2000. Wie in Kapitel 4.1 beschrieben, und auch von der ADL untersucht, existieren aktuell Trends im eLearning, die völlig andere Anforderungen an eLearning-Inhalte stellen als zur Zeit der Entwicklung von SCORM. Viele neue Trends entwickelten sich aus weitaus jüngeren Technologien als die, mit denen SCORM umgesetzt wurde, womit einige der aktuellen Anforderungen mit SCORM überhaupt nicht oder nur schwer realisierbar sind (mobiles Lernen, Nutzung von sozialen Netzwerken, …). Die Anforderungen an einen Standard für „moderne“ WBTs sind somit durchaus andere, als zur Zeit der Entwicklung von SCORM.

Mit Hilfe der *Training and Learning Architecture* (im Folgenden auch TLA genannt), möchte die ADL eine zeitgemäße Architektur für eLearning-Inhalte entwickeln. Mit der TLA sollen alle aktuellen Anforderungen des eLearnings (darunter auch die in Kapitel 4.1 vorgestellten aktuellen Trends) umgesetzt werden können. Die TLA soll somit standardisierte Lösungen für aktuelle Anforderungen des eLearnings bieten.

Wie bei SCORM handelt es sich bei der TLA um einen Überbegriff für verschiedene Komponenten, die das Erzeugen von eLearning-Inhalten beschreiben und vereinfachen sollen. Arbeitstitel des Projekts war „The Next Generation of SCORM“, was die Intention des Projekts wohl am besten definiert.

Dabei soll die TLA SCORM nicht komplett ersetzen, sondern eine neues Gesamtkonzept anbieten, mit dem aktuelle Anforderungen an eLearning-Inhalte realisiert werden können. Einzelne Teile von SCORM können dabei durchaus weiterhin, als Teil der TLA, Verwendung finden.

4.2.1 LETSI

Grundlage für die Arbeiten an der TLA lieferte primär das 2008 gestartete Projekt „SCORM 2.0“ der *International Federation for* L*earning,* E*ducation, and* T*raining* S*ystems* I*nteroperability* (kurz *LETSI*).

Im Projekt „SCORM 2.0“ startete die LETSI 2008 einen Aufruf an alle eLearning-Interessierten sowie Autoren und Firmen. Die LETSI forderte dazu auf, Whitepaper mit Ideen, Vorschlägen und prototypischen Umsetzungen einzureichen wie sich die Beteiligten ein „SCORM 2.0“ vorstellen würden.

Es gingen über 100 Whitepaper ein, die auf [LE08] eingesehen werden können.

Aufbauend auf diesen Ideen zur Erweiterung von SCORM konnte die LETSI die Hauptkritikpunkte an SCORM ablesen und begann mit prototypischen Umsetzungen, um SCORM den neuen Bedürfnissen der Autoren anzupassen.

Ein daraus entstandenes Projekt ist der *LETSI RTWS* (Run-Time Web Service), der das Run-Time Modell von SCORM durch einen WebService auf SOAP-Basis ersetzt. Ziel des LETSI RTWS war es das LMS, das bei SCORM immer im Hintergrund vorhanden sein muss, obsolet zu machen. WBTs sollten auch außerhalb eines LMS gestartet werden

können und trotzdem die anfallenden Daten über den SOAP-WebService an das LMS melden. Allerdings kam das Projekt nie über den prototypischen Status hinaus und fand bislang noch keine große Verbreitung. [12]

4.2.2 Hauptkritikpunkte an SCORM

Ausgehend von den eingereichten Whitepapers sowie weiterer Quellen (von der ADL geführten Interviews mit verschiedenen Firmen aus dem Bereich eLearning, verschiedenen „User Voice“ Foren im Internet und den Ergebnissen verschiedener Arbeitsgruppen der ADL) lassen sich folgende Hauptkritikpunkte an dem Konzept von SCORM (1.2 sowie 2004) und somit Anforderungen an das „neue SCORM“, zusammenfassen[13]:

Unterstützung externer SCORM-Inhalte/WBTs

SCORM setzt für alle Funktionen, die in der Spezifikation beschrieben werden, ein LMS als Distributions-Plattform für die SCORM-Inhalte voraus. Dies führt zu mehreren Anwendungsfällen, die mit SCORM nicht (oder nur sehr schwer) realisierbar sind.

a) SCORM-Inhalte sollen auch außerhalb eines LMS möglich sein

Hauptziel während der Entwicklung von SCORM war es, die Interoperabilität und Wiederverwendbarkeit von eLearning-Inhalten zu erhöhen. Zur Umsetzung dieser Ziele setzte SCORM voll und ganz auf die Verwendung von LMS zur Verbreitung und Verwaltung von Lerninhalten.

Daher bietet SCORM keinerlei Möglichkeiten, Inhalte, die außerhalb des LMS liegen, zu erfassen und zu verwerten. Heutzutage sind allerdings Techniken wie Cloud-Computing oder allgemein die Dezentralisierung von Content geläufig. So schätzen Autoren des Web 2.0 die Möglichkeit, Inhalte einmalig bereitzustellen und danach durch Verlinkung zu veröffentlichen (wie auch in 4.1.1 MOOCs beschrieben). Die Autoren behalten dabei die volle Kontrolle über die veröffentlichten Inhalte, was wichtig ist, will man die Inhalte überarbeiten oder auch zur Einhaltung und Kontrolle von Rechtsfragen, die heutzutage im Internet immer bedeutsamer werden.

Die Distribution eines WBTs auf einem anderen Server als dem LMS-eigenen ist von SCORM nicht vorgesehen. Durch die JavaScript-basierte Run-Time Environment ist die API den Regeln des *Cross-Domain Scriptings* unterworfen. Diese Sicherheitsregeln verbieten es im Allgemeinen, dass JavaScript-Code, der nicht auf demselben Server liegt wie die aufgerufene Seite (das LMS), ausgeführt werden kann.

Die ADL hat ein Paper zu dieser Thematik veröffentlicht, in dem einige Lösungsmöglichkeiten für dieses Problem genannt werden [Ad03]. Diese Lösungen sind

[12] Siehe Abbildung 2
[13] Vgl. [Ad12d]

allerdings ziemlich umständlich und fallen teilweise fast unter die Kategorie „Hacks“, da sie verschiedene browserspezifische Lücken ausnutzen, um den extern gehosteten JavaScript-Code ausführen zu können. Das Verbieten von *Cross-Domain Scripting* ist eine Sicherheitsregel von JavaScript, die durchaus Sinn macht, um Internet-Nutzer vor Angreifern zu schützen, die Schadcode ausführen wollen. Ein Umgehen dieser Regel ist also nicht nur äußerst fragwürdig, sondern führt auch zu einer geringeren Browserkompatibilität der WBTs, die diese Regel zu umgehen versuchen.

b) Auch nicht-SCORM Inhalte sollen protokollierbar sein

Das gegenwärtige Lernen konzentriert sich nicht mehr nur auf die von Lehrenden bereitgestellten Inhalte im LMS, sondern die Lernenden vernetzen sich zunehmend in externen Social-Media-Plattformen, um sich über das Gelernte auszutauschen oder suchen auf anderen Plattformen wie Wikipedia nach weiterführenden Informationen. All diese zusätzlichen Angebote kann ein LMS gar nicht leisten und demzufolge wäre es wünschenswert, wenn man Aktivitäten von Lernenden in externen Tools genauso erfassen könnte, wie im LMS getätigte Aktivitäten. Dadurch könnte man ein viel genaueres Lernprotokoll für die Lernenden anlegen und deren Fortschritte nicht nur an vereinzelten, für SCORM aufbereiteten, Inhalten messen.

So sollte den Lernenden die Möglichkeit gegeben werden, das Lesen eines nützlichen Wikipedia-Artikels an das LMS (und somit den Lehrenden) zu melden oder auf einen selbst geschriebenen Blog-Beitrag zu verweisen.

Auch solche, nicht in das abgekapselte System eines WBTs integrierbaren Aktivitäten, sollten erfassbar gemacht werden. Das „neue SCORM“ soll also nicht nur in WBTs verwendet werden können, sondern Web-Inhalte aller Art protokollieren können.

c) SCORM sollte auch browserunabhängig funktionieren

Serious Games, Simulationen oder andere komplexe Lernanwendungen benötigen teilweise Technologien, die in einem Browser nicht verfügbar sind. Bislang gibt es kaum Möglichkeiten, Aktivitäten in anderen Programmen als dem Browser zu erfassen. Der Markt der Serious Games oder der virtuellen Realitäten, wie zum Beispiel „Second World“, kann nicht mit SCORM bedient werden.[14]

Auch bei der mobilen App- oder WBT-Entwicklung gibt es mit SCORM Einschränkungen, die man beachten muss. WBTs, die als Web-Apps entwickelt wurden, sind zwar möglich, hybride- oder native Umsetzungen von WBTs sind allerdings fast ausgeschlossen.

Bei Web-Apps, die SCORM-Funktionalitäten nutzen sollen, muss der Lernende das WBT immer über das LMS aufrufen. Für das LMS bedeutet dies, dass es im besten Fall eine Version, speziell für mobile Geräte, bereitstellen sollte. Außerdem muss das LMS beim Bearbeiten des WBTs immer geöffnet bleiben. Ein Anzeigen des WBTs in einem iFrame

[14] Für die Spiele-Engine *Unity* existiert ein SCORM-PlugIn [Ad12c], welches es ermöglicht mit Unity entwickelte Browser-Spiele, in ein LMS einzufügen und Ergebnisse mit SCORM zu tracken.

innerhalb des LMS ist auf mobilen Geräten als kritisch anzusehen, da durch die verhältnismäßig kleinen Displays auf Smartphones jeder verfügbare Platz für das WBT verwendet und das LMS somit im besten Fall ausgeblendet werden sollte. Auch ein Starten als PopUp ist als kritisch anzusehen. Die meisten Smartphones unterstützen zwar die Möglichkeit mehrere Webseiten parallel in Tabs anzuzeigen, jedoch könnte die Performance des WBTs darunter leiden, dass das LMS ständig im Hintergrund bereitgehalten werden muss.

Der Einsatz von nativen und hybriden Apps ist dagegen nur sehr eingeschränkt möglich. So kann man nicht direkt das WBT oder den Lerninhalt als App ausliefern, sondern man müsste sich darauf beschränken, mit der App einen systemeigenen Browser zu öffnen, in dem man dann die Webseite des LMS aufruft. Im LMS muss das WBT dann als Web-App bereitgestellt sein, sodass es anschließend aufgerufen werden kann (mit den oben beschriebenen Einschränkungen der Web-Apps).

Da das LMS immer im Hintergrund vorhanden sein muss und die Daten auch von keinem anderen Standort kommen dürfen als dem LMS-Server, gibt es keine Möglichkeit ein Stand-Alone-WBT mit SCORM-Unterstützung zu entwickeln. Das „neue SCORM“ sollte diese Möglichkeit auf jeden Fall bieten und die Beschränkung, einen Browser verwenden zu müssen, aufheben.

d) SCORM sollte auch offline Phasen beachten und Lösungen dafür anbieten

Eine weitere Einschränkung zur Implementierung von SCORM in Serious Games oder mobilen WBTs ist, dass eine ständige Online-Verbindung zur Nutzung von SCORM erforderlich ist. Die Verbindung zum LMS muss jederzeit verfügbar sein, ansonsten laufen die Daten-Anfragen ins Leere und die SCORM-API des LMS beendet den aktuellen Versuch.

Trotz der steigenden Verbreitung von Datenverbindungen auf Smartphones muss man bei mobilen Geräten davon ausgehen, dass die Verbindung zum Server nicht so stabil ist wie bei Desktop-PCs mit festem Standort. Fährt der Nutzer mit dem Zug durch einen Tunnel ist die Verbindung zum LMS unterbrochen, das LMS kann keine Daten mehr empfangen und die API-Verbindung wird beendet.

Auch bei Serious Games muss man darauf achten, da die Lerneinheiten hier womöglich sehr viel länger sind als bei klassischen WBTs und die Wahrscheinlichkeit für eine Unterbrechung der Internet-Verbindung während der Ausführung damit steigt.

➔ Das SCORM-RTE muss grundlegend überarbeitet werden, um externe Inhalte inklusive Datenerfassung möglich zu machen.

Die SCORM-Spezifikationen sind zu komplex

Die vier Bücher der SCORM 2004 Spezifikation umfassen über 1000 Seiten. Einerseits sind dadurch alle Funktionen sehr ausführlich und präzise definiert, auf der anderen Seite engt genau diese präzise Definition den Autor in seiner Kurs-Gestaltung sehr ein.

Außerdem könnte es sein, dass die sehr umfangreiche Spezifikation neue Autoren, die SCORM verwenden möchten, abschreckt.

➔ Das neue SCORM muss einfacher werden!

Die Aussagekraft der von SCORM akzeptierten Daten ist nicht ausreichend für komplexer werdende Web-Inhalte

Durch die Verwendung des CMI-Data Models in der SCORM Run-Time Environment ist die Menge der von SCORM unterstützten/akzeptierten Daten sehr eingeschränkt. Die übermittelten Daten müssen genau der Spezifikation folgen und es ist nicht immer möglich, alle anfallenden Daten eines WBTs zu erfassen.

So ist zum Beispiel die Größenbeschränkung des Daten-Feldes `cmi.suspend_data` in SCORM 1.2 auf 4.096 Zeichen und in SCORM 2004 auf 64.000 Zeichen beschränkt. Nichtsdestotrotz ist dieses Datenfeld, das für das Abspeichern des aktuellen Kurs-Zustandes zur Verfügung steht, allein durch diese Einschränkung nicht mehr zeitgemäß. Web-Inhalte und damit auch WBTs werden zunehmend komplexer und somit stößt man immer öfter an die Grenzen der durch SCORM definierten Datenfelder.

Auch die Aussagekraft der Daten ist massiv eingeschränkt. So gibt es zum Beispiel keine Möglichkeit das Ansehen eines Videos oder das Folgen eines externen Internet-Links zu protokollieren. Für jegliche Aktivitäten des Lernenden, die nichts mit dem Beantworten einer Frage gemein haben, gibt es keine passenden Strukturen im SCORM Data-Model (weder in SCORM 1.2, noch in SCORM 2004).

➔ Die Möglichkeiten der Datenerfassung müssen flexibler werden, um die Aussagekraft der Daten zu erhöhen.

Die erfassten Daten sollten besser weiterverwendbar/abrufbar sein

SCORM beschreibt, wie die Daten zum LMS gesendet werden. Was aber danach mit den Daten geschieht, bleibt dem LMS überlassen. Sind die Daten einmal im LMS angekommen, so gibt es nur Regeln, dass das SCORM-Paket, das die Daten gesendet hat, Teile der Daten auch wieder anfordern kann. Weitere Vorschriften, wie das LMS mit den Daten umzugehen hat und noch wichtiger, welche Daten es dem Kurs-Autor anzeigen soll, oder ob dem Autor irgendeine Möglichkeit zum Export der Daten gegeben werden sollte, sind nicht vorhanden.

Es existieren zwar LMS mit umfangreichen Möglichkeiten zur Bewertung von Nutzern und manchmal sogar zur Bewertung von Gruppen oder der Definition von Lernzielen und ähnlichem (zum Beispiel Moodle), allerdings sind dies alles Dinge, die SCORM bislang nicht definierte. Viele LMS nehmen die Daten zwar an, danach sind sie allerdings für den Autor nur schwer aus- und verwertbar.

➔ Der Umgang mit den erfassten Daten muss offener gestaltet werden, sodass zum Beispiel externe Reporting-Tools darauf zugreifen können.

Sequencing and Navigation sollte aus SCORM entfernt werden

Die in SCORM 2004 eingeführten Regeln für *Sequencing and Navigation (S&N)*, die es ermöglichen Ablaufpläne und Beschränkungen zur Navigation zwischen einzelnen SCOs in einem SCORM-Paket zu erstellen, werden von der Mehrheit der SCORM Autoren nicht verwendet.

Zu diesem Schluss ist die ADL nach Auswertung einzelner Statistiken zum Thema gekommen. So kann man in den Statistiken der SCORM Cloud (ein von Rustici Software angebotenes Web-Portal zum Speichern von SCORM-Inhalten) erkennen, dass über die Hälfte aller hochgeladenen SCORM-Pakete noch SCORM 1.2 anstatt der neueren Version 2004 verwenden. Von den unter 50% der SCORM 2004 Pakete verwenden wiederum nur ungefähr 50% eine der S&N Erweiterungen. Somit verwenden nach den von Rustici Software bereitgestellten Daten nur circa ein Viertel aller SCORM-Pakete S&N.

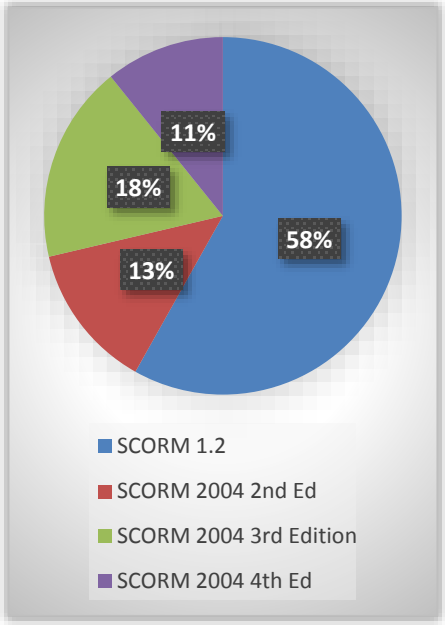


Abbildung 12 - Verwendete SCORM Version aller 48.000 hochgeladenen SCORM-Pakete unter http://cloud.scorm.com/

Primärer Grund ist wohl, dass SCORM 2004 außer S&N keine neuen Funktionalitäten mitbringt. Die weiteren Änderungen sind, wie schon in Kap. 3.5.1.2 erwähnt, marginal.

SCORM 2004 Edition	Use Rollup Rules	Use Sequencing Rules	Use Global Objectives	Use Any Sequencing
2nd Edition	10%	11%	6%	17%
3rd Edition	28%	33%	31%	58%
4th Edition	45%	19%	14%	54%
Any Edition	27%	25%	21%	47%

Abbildung 13 – Verwendung der von Sequencing and Navigation eingeführten Regeln, basierend auf den Daten aller 48.000 auf http://cloud.scorm.com/ hochgeladenen SCORM-Pakete.

Ein weiterer Punkt ist, dass die Umsetzung von S&N für Autoren sowie für LMS-Entwickler sehr komplex ist, sodass viele LMS darauf verzichten und SCORM nur in Version 1.2 unterstützen. Somit verwenden die meisten Autoren, die S&N nicht benötigen, meist einfach SCORM 1.2.

➔ Verwerfen der Sequencing & Navigation-Erweiterungen

Der Lernende sollte nicht mehr nur als einzelne Person behandelt werden

In Anbetracht der steigenden Vernetzung der Lernenden und dem Fakt, dass in SCORM alle Daten immer nur für einen einzelnen Lerner gelten, sollte es möglich sein, auch Lerngruppen (Team-Based-Training) anzugeben.

➔ Das SCORM Data-Model erweitern (Multi-User Tracking)

4.2.3 Anforderungen an einen neuen eLearning Standard

Zusammenfassend sind die zentralen Anforderungen, die die ADL nach Analyse der Hauptkritikpunkte an SCORM an einen neuen eLearning-Standard stellt:

- Das SCORM-RTE muss grundlegend überarbeitet werden um externe Inhalte inklusive Datenerfassung möglich zu machen.
- Die Möglichkeiten der Datenerfassung müssen flexibler werden, um die Aussagekraft der Daten zu erhöhen.
- Der Umgang mit den erfassten Daten muss offener gestaltet werden, sodass zum Beispiel externe Reporting-Tools darauf zugreifen können.
- Das SCORM Data-Model um Multi-User Tracking erweitern.
- Die Sequencing & Navigation-Erweiterung muss verworfen werden.

Die ADL fasst diese Anforderungen in zwei Aufgaben für die Entwicklung eines neuen eLearning-Standards zusammen:

„Make it simple…

– Make it powerful…“ [Ad12b]

Vergleicht man die Anforderungen an eLearning, die aus den Trends des *Horizon Reports* in Kapitel 4.1 hervorgegangen sind, mit den Anforderungen, die die ADL sich für die Entwicklung der TLA gesetzt hat, so ähneln sich beide sehr.

Beide haben das Hauptziel, Lerninhalte nicht mehr nur in LMS zu verankern, sondern auch externe Plattformen zum Lernprozess hinzuzufügen. Somit sollen in Zukunft nicht mehr nur klassische WBTs als zentrales Mittel zur Vermittlung von Lerninhalten verwendet werden, sondern auch die Möglichkeiten, die das Web 2.0 mit den verfügbaren *Social-Media Plattformen* bietet, integrierbar werden. Außerdem sollen *Out-Of-Browser-Experiences* möglich werden, die die Verwendung alternativer Geräte und Umsetzungen von Lerninhalten möglich machen sollen.

4.2.4 Die Training and Learning Architecture (TLA)

Auf Basis dieser zeitgemäßen Anforderungen an das eLearning entstand das Konzept der *Training and Learning Architecture (TLA).*

Die TLA besteht aus vier Komponenten, deren Hauptziel eine moderne Spezifikation für die plattformübergreifende Verwaltung und Kategorisierung von Lernenden und Lernobjekten sowie die Erfassung der von Lernenden getätigten Aktivitäten ist.

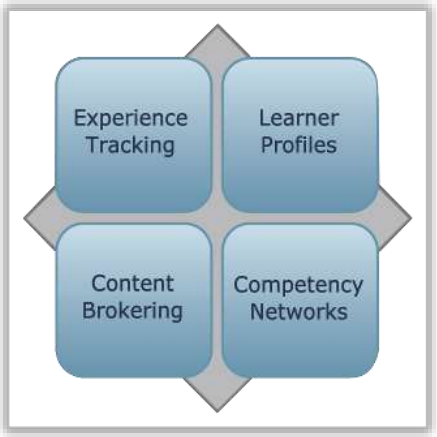


Abbildung 14 – Komponenten der TLA

Zum Zeitpunkt der Umsetzung dieser Arbeit (Juli/August 2013) befand sich das Projekt TLA noch in Phase 1. Somit sind alle Komponenten, bis auf das Experience Tracking, noch nicht endgültig definiert und genauere Spezifikationen noch nicht vorhanden, wodurch konkrete Informationen in dieser Phase nur schwer zu finden waren. Die folgenden Angaben können sich somit noch ändern und sollen nur das Grundkonzept beschreiben.

Die vier Komponenten der TLA:

Experience Tracking:

Das Experience Tracking beschreibt die technische Grundlage des TLA und die Art und Weise, wie Aktivitäten der Lernenden protokolliert (getracked) werden und wie anschließend mit den Daten umgegangen wird.

Umgesetzt wurde dies mit dem *Project TinCan* und der daraus entstandenen *Experience API* (siehe folgenden Abschnitt).

Man kann das Experience Tracking somit als Gegenstück zur SCORM Run-Time Environment sehen. Es beschreibt die Form der übermittelten Daten, sowie die Technologien mit denen dies umgesetzt wird.

Learner Profiles:

Für jeden Lernenden soll es die Möglichkeit geben, ein Portfolio zu erstellen, in dem alle Lernerfolge (oder auch Misserfolge) sowie vom Lerner selbst eingetragen Daten wie Vorlieben oder Vorkenntnisse verzeichnet sind. Anhand dieses Portfolios kann der Lernende einerseits kursübergreifend bewertet werden, andererseits sollen die im Learner-Profile bereitgestellten Informationen auch dazu dienen, dem Lernenden maßgeschneiderte Lernerfahrungen anzubieten. Weiß man zum Beispiel schon, dass der Lernende ein gutes Mathematikverständnis hat, so kann man im nächsten Kurs, an dem er teilnimmt, die Einführung in Mathematik vielleicht abkürzen. Man kann somit die Learner Profiles als eine Möglichkeit zur Darstellung des *Life-Long Learners* sehen. Ebenso ist vorstellbar, dass die in MOOCs erarbeiteten Badges mit Hilfe von Learner Profiles verwaltet und gesammelt werden können.

Ein zentraler Punkt der Spezifikation der Learner Profiles wird auch der Umgang mit den Daten sein. Wie authentifiziert sich der Nutzer? Wer darf auf die gespeicherten Daten zugreifen? Diese und andere Fragen müssen allerdings noch geklärt werden.

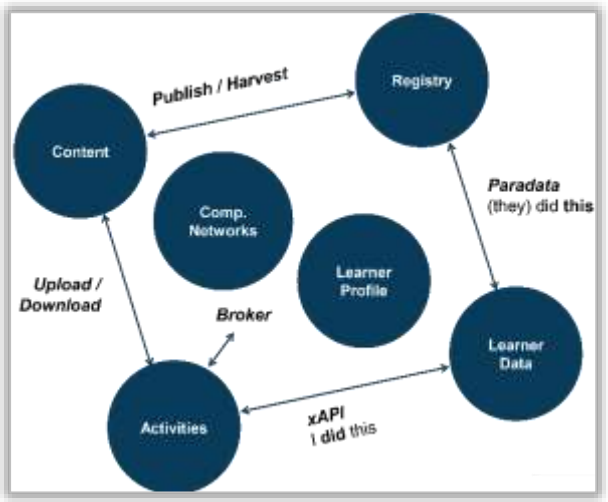


Abbildung 15 – Zusammenhang der Komponenten der TLA

Content Brokering:

Content Broker sollen dazu in der Lage sein, verschiedene Lerninhalte aus unterschiedlichen Lernobjektdatenbanken zu organisieren und anhand ihrer angegebenen Metadaten zu analysieren. Auf Basis dieser Daten sollen Lernobjekte vom Content Broker so ausgewählt und kombiniert werden, dass dem Lernenden das für ihn passendste, nächste Lernobjekt angeboten wird.

Content Broker sollen somit das Bindeglied zwischen Lernobjektdatenbanken und den Learner Profiles sowie den Competency Networks sein, um die dem Lernenden angebotenen Inhalte in einer sinnvollen Reihenfolge zu kombinieren und anzubieten.

Competency Networks:

Mit Competency Networks (Kompetenznetzwerken) sollen Kompetenzen von Lernenden sowie von Lernobjekten definiert und verwaltet werden können. So können einerseits Lernenden Kompetenzen zugewiesen werden, die sie durch Bearbeitung unterschiedlicher Lerninhalte erlangen sollen, andererseits können den Lerninhalten Informationen zugewiesen werden, welche Kompetenzen mit ihnen gefördert werden.

Kompetenznetzwerke arbeiten somit mit den Learner Profiles und den Content Brokern zusammen, um den Lernenden die für sie passenden Lerneinheiten je nach Anforderungsprofil anbieten zu können.

4.3 Die Experience API

Die Entwicklung der TLA läuft phasenweise ab. Die erste Phase ist die Umsetzung und Spezifikation des *Experience Trackings.*

Die *Experience API* (im Folgenden auch xAPI genannt) stellt die Umsetzung des *Experience Trackings* dar und ist somit zentraler Bestandteil und technische Basis der *Training and Learning Architecture* (*TLA*).

Die xAPI soll es ermöglichen, alle von der TLA benötigten Daten durch ein geeignetes Daten-Modell abzubilden. Ebenso muss sie eine moderne Schnittstelle zur Kommunikation mit dem Datenspeicher bereitstellen, um die Daten abspeichern und wieder abrufen zu können.

Als Einführung in die Experience API wird zunächst die Entwicklung der xAPI sowie ihre Position in der TLA beschrieben. Anschließend werden die Hauptanforderungen in der Konzeption der xAPI definiert und abschließend wird das daraus entwickelte Grundkonzept der xAPI vorgestellt.

4.3.1 Entwicklungsphasen der Experience API

Die Entwicklung der Experience API lief in unterschiedlichen Phasen ab.

2008 - LETSI Projekt SCORM 2.0

Wie in Kapitel 4.2.1 beschrieben, startete die LETSI 2008 einen Aufruf zur Einreichung von Ideen für ein zukünftiges SCORM 2.0. Über 100 Whitepaper wurden eingereicht und ausgewertet. Dieser Schritt gehört noch nicht zur geplanten Entwicklung der xAPI, bildet jedoch eine der Grundlagen der Anforderungen.

2010 – ADL - Broad Agency Announcement (BAA)

Anhand der LETSI-Whitepapers entwirft die ADL das Grundkonzept für die TLA und veröffentlicht eine Ausschreibung (BAA) für die Spezifikation und technische Umsetzung des *Experience Trackings*.

Rustici Software, die die ADL auch schon bei der Entwicklung von SCORM 2004 unterstützt hat, bekommt den Auftrag zugesprochen.

2010 – 2011 – Project TinCan

Rustici Software startet das *Project TinCan*. Dabei werden als erstes die LETSI Whitepaper analysiert und weitere Stimmen aus der eLearning-Industrie angehört. Weiterhin werden zahlreiche „User Voice“ Foren im Internet mit wöchentlichen Meetings und vereinzelt stattfindenden realen Treffen eingerichtet.

Project TinCan ist als *Community-Driven Project* angelegt worden und setzt auf die Beteiligung der eLearning-Community. Ebenso werden erarbeitete Spezifikationen öffentlich verfügbar gemacht, um die Reaktionen der Community in späteren Versionen berücksichtigen zu können.

Aus den gesammelten Beiträgen der Community sowie der LETSI Whitepapers wird ein Anforderungskatalog für Project TinCan erstellt.[15]

2012 – TinCan API

Zur Vorstellung auf der MLearnCon12 wurde im Juni 2012 die Version 0.9 der Spezifikation der TinCan API veröffentlicht und der eLearning-Community vorgestellt.

Aus dem Project TinCan wurde somit die TinCan API. 15 *Early-Adaptors* hatten die TinCan API zu diesem Zeitpunkt bereits in ihre Produkte eingebunden.

2012 – Experience API

Im August 2012 folgte Version 0.95 der TinCan API - Spezifikation. Die ADL sah die von der Ausschreibung geforderten Anforderungen an eine Umsetzung des *Experience Trackings* als erfüllt an und übernahm die von Rustici Software entwickelten Spezifikationen. Die aus dem Project TinCan entstandene TinCan API wurde unter dem Namen *Experience API* in das Konzept der TLA eingebunden und stellt die technische Lösung des *Experience Trackings* dar.

[15] Siehe Kapitel 4.2.3 - Anforderungen an einen neuen eLearning Standard
- Alle verwendeten Logos von den jeweiligen Webseiten

2013 – Experience API v1.0

Am 26. April 2013 erschien Version 1.0 der Spezifikation der Experience API.

Die ADL hatte die Spezifikationen von Rustici Software übernommen und überarbeitet. Seit der Übernahme durch die ADL ist der offizielle Name also Experience API. Jedoch ist durch die offene Entwicklung und die Community-Arbeit seitens Rustici Software der Begriff TinCan API immer noch geläufig. Rustici Software bietet außerdem unter dem Namen TinCan API ihre eigene Implementierung der Experience API kommerziell an. Die Experience API ist somit der Name der technischen Spezifikation, die TinCan API stellt jedoch eine fertige Implementierung dieser Spezifikation dar.

Zukünftige Schritte

Die ADL möchte im Laufe des Jahres 2013 die Spezifikationen der Experience API an ein Standard-Gremium übergeben, um von diesem die Experience API als einen neuen eLearning-Standard anerkennen zu lassen.

4.3.2 Ziele der Experience API

Die ADL beschreibt einen Lernvorgang als einen vom Lernenden ausgehenden Prozess.

Abbildung 16 – Ziele der Experience API

Dabei verwendet der Lernende verschiedene Werkzeuge (Tools) wie Computer, Smartphones, WBTs, soziale Plattformen oder auch Offline-Medien wie Bücher zum Lernen.

Mit diesen Tools werden vom Lerner bestimmte Aktivitäten (*Activities*) durchgeführt. So kann der Lernende ein Video ansehen, ein WBT bearbeiten, einen Blogbeitrag bewerten oder ein Buch lesen.

Lernen geschieht also überall und mit den verschiedensten Werkzeugen. Mit diesen Werkzeugen führt der Lernende Aktivitäten aus, die zum Lernprozess beitragen.

Alle diese getätigten Aktivitäten sollen auf irgendeine Art erfasst werden können. Außerdem sollen Möglichkeiten bereitgestellt werden, dass die Daten anschließend verarbeitet, ausgewertet und wiederverwendet werden können.

Mit der Experience API möchte die ADL somit eine API entwickeln, die es möglich macht:

„geräte- und systemübergreifend Erfahrungen aller Art zu protokollieren und verfügbar zu machen“

Daraus kann man folgende Ziele festhalten, die die Experience API erreichen muss, um die TLA zu einer Architektur zu machen, die die Anforderungen des aktuellen eLearnings erfüllt:

Geräte- und Systemunabhängigkeit

Die API soll geräte- und systemunabhängig eingesetzt werden können.

- ➔ *Aktivitäten sollen unabhängig davon auf welchem Gerät und mit welcher Technik sie umgesetzt wurden, erfassbar sein.*
- ➔ *mobile sowie neuartige Geräte wie SetTop-Boxen für Fernseher können unterstützt werden.*
- ➔ *Out-Of-Browser-Experiences werden möglich. (Serious Games, Simulationen oder auch Offline-Erfahrungen)*

Flexibles Daten-Modell

Das Daten-Modell, das die Form der von der API akzeptierten Daten beschreibt, sollte so offen und flexibel wie möglich gestaltet werden.

- ➔ *Die Aktivitäten und Erfahrungen, die protokolliert werden können, sollen nicht durch zu strenge Definitionen eingeschränkt werden.*
- ➔ *Jegliche Art der Aktivität soll erfassbar werden. (Videos ansehen, Kurse abschließen, Bücher lesen, andere Personen treffen, …)*

Verfügbarkeit der Daten

Die API sollte die erfassten Daten nach ihrer Erfassung nach außen verfügbar machen.

- ➔ *Die gesammelten Daten über die gemachten Erfahrungen sollen nach der Erfassung wiederverwendbar sein.*
- ➔ *Verschiedene Systeme sollen die erfassten Daten anfordern und verwenden können.*

Zusammenfassend kann man somit feststellen, dass die Experience API, sofern Sie alle Ziele erreichen kann, eine geeignete Basis-Technologie für die Training and Learning Architecture darstellt.

Alle Aspekte der technischen Anforderungen der TLA werden durch die Zielsetzungen der Experience API beachtet und scheinen mit Hilfe dieser Definition der Experience API umsetzbar.

Somit könnte auch die Zielsetzung der ADL, mit der TLA Lösungen für die aktuellen Anforderungen des eLearnings anbieten zu wollen, mit der Experience API als Basis erfolgreich umgesetzt werden.

All dies setzt jedoch voraus, dass die Umsetzung der Experience API alle an sie gestellten Anforderungen erfüllen kann. Zu diesem Zweck wird nun die Umsetzung der Experience API in ihren Grundzügen vorgestellt und anschließend in *Kapitel 5 – Analyse der Spezifikation der Experience API* genauer untersucht.

4.3.3 Bestandteile der Experience API

Um eine technische Umsetzung aller drei gestellten Anforderungen zu realisieren, werden zwei zentrale Komponenten für die Experience API eingeführt.

Die *Learning Record Stores*, die als WebServer die Daten empfangen und verteilen können sowie die *Statements*, die das verwendete Daten-Modell beschreiben.

4.3.3.1 LRS – Learning Record Store

Um das Ziel der Geräte- und Systemunabhängigkeit sowie der von überall aus verfügbaren Schnittstelle zum Austausch von Daten zu erreichen, kann (wie in Kapitel 4.2.2 - *Hauptkritikpunkte an SCORM* beschrieben) keine Technik in Form der SCORM Run-Time Environment verwendet werden.

Im SCORM-RTE wird vom Lerninhalt aus eine direkte Verbindung zu der, vom LMS bereitgestellten, ECMAScript-basierten API aufgebaut. Diese direkte Verbindung, die bedingt, dass sich der Client (Lerninhalt) sowie der Server (das LMS) im gleichen Browser befinden müssen, ist nicht dazu geeignet, um eine systemunabhängige Verbindung zu realisieren.

Flexibler ist dabei die Verwendung eines WebServices, der immer unter einer festen URL im Internet erreichbar ist, und auf Anfragen des Clients hin reagiert.

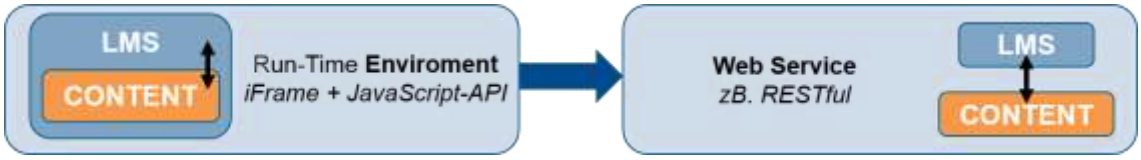


Abbildung 17 - SCORM RTE vs. REST-WebService

Der Learning Record Store (LRS) ist ein solcher REST-basierter WebService, der dazu dienen soll, die Daten, die bei Verwendung eines Lerninhaltes anfallen, zu speichern und ebenso wieder abrufbar zu machen.

In der Experience API Spezifikation werden Inhalte, die Daten an das LRS bereitstellen (wie zum Beispiel WBTs, einzelne Webseiten, Spiele oder Simulationen), *Learning*

Activity Providers (*APs*) genannt. Die übertragenen Daten werden als *Statements* bezeichnet, da diese Aussagen über getätigte Aktivitäten des Lernenden darstellen.

Die *Activity Provider* senden die anfallenden *Statements* an das LRS, das diese entgegennimmt und abspeichert. Andere APs sowie andere LRS oder Services wie Auswertungs-Tools, können anschließend gezielt Daten vom LRS anfordern, sofern sie das Recht dazu haben.

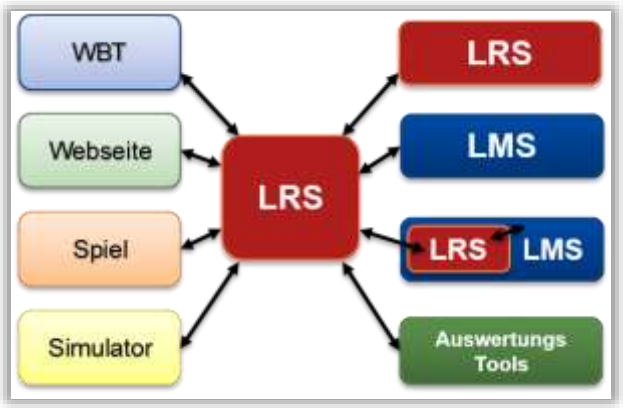


Abbildung 18 – Datenaustausch mit dem LRS

Das LRS steht in der Umsetzung der Experience API also im Mittelpunkt und wird als zentrale Anlaufstelle zur Speicherung und Verteilung der anfallenden Statements verwendet.

Das LRS übernimmt somit weitestgehend Funktionen von herkömmlichen LMS, indem es die Datenverwaltung übernimmt. Dabei kann ein LRS allerdings auch direkt in ein LMS eingebunden werden. Alternativ fordert das LMS die Daten, wie alle anderen Services auch, vom LRS an.

Zusätzlich können APs mehrere LRS gleichzeitig ansprechen, falls dies gewünscht ist. Zum Beispiel wäre denkbar, dass ein Trainings-WBT in einer Firma die Statements gleichzeitig an ein Haupt-LRS sowie an ein Abteilungs-LRS sendet. Alternativ könnte das Haupt-LRS auch die Daten von mehreren Abteilungs-LRS anfordern. Wie dies umgesetzt wird, muss durch die Struktur des Gesamtsystems entschieden werden.

➔ Durch die Umsetzung des LRS als RESTful-WebService kann eine Loslösung der Daten aus dem LMS erreicht werden. Die Besonderheit eines RESTful-WebServices ist, dass er von jedem Programm angesprochen werden kann, das in der Lage ist, http-Requests zu senden (vereinfacht: Webseiten aufrufen kann). Somit könnte damit das Ziel eines **system- und geräteübergreifenden Zugriffs** erreicht werden.

➔ Durch die Definition, dass ein LRS nicht nur Daten empfangen, sondern ebenso auch wieder senden kann, kann auch das Ziel der **Verfügbarkeit der Daten** erreicht werden.

4.3.3.2 Statements

Wie bereits beschrieben, werden die Daten, die von den *Activity Providern* (APs) gesendet werden, in Form von *Statements* angegeben. Da die Anforderung an das Daten-Modell war, es so flexibel wie möglich zu machen, um möglichst alle Aktivitäten des Lernenden abbilden zu können, stellen die Statements einzelne, abgeschlossene Aussagen über die jeweilige Aktivität dar.

Die Form der Statements ist dabei an die Form von *Activity Streams* angelehnt.

Activity Streams

Activity Streams sind von Plattformen wie Facebook, Google+ oder Twitter bekannt. So stellt die Facebook Pinnwand oder das persönliche Aktivitätenprotokoll einen *Activity Stream* dar.

Auf diesen Seiten kann der Nutzer seine oder die Aktivitäten anderer im zeitlichen Ablauf und für Menschen gut lesbar nachverfolgen. Beispiele dafür wären:

- „Peter hat SpielX gespielt“
- „Susi hat Klaus eine private Nachricht geschrieben“
- „Patrick hat eine Seite geteilt: ExperienceAPI-Documentation“

Die einzelnen Aktivitäten sind dabei Statements, die eine eindeutige Aussage über eine bestimmte Aktivität treffen und die zusammen einen Activity Stream ergeben.

Alle Statements haben eine bestimmte Form, so folgen sie (im Englischen) immer dem Prinzip: „I did this“. Also sind Objekt, Verb und Subjekt die Minimalanforderungen eines Statements.

Es existieren momentan Bestrebungen die Form der Activity Streams zu standardisieren, um diese besser plattformübergreifend nutzen zu können[16].

Form der Statements

Wie bei den Acitity Streams folgen einzelne Statements in der Experience API der Form „I did this“ und werden im JSON-Format abgebildet und an das LRS geschickt.

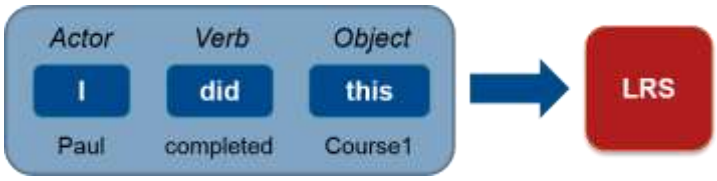


Abbildung 19 - Einfaches Statement der Form „I did this“

Der Akteur der Aktivität wird dabei in der Experience API als „Actor“ bezeichnet und kann nicht nur eine einzelne Person sein sondern auch eine Gruppe von Akteuren.

Das Verb beschreibt wie im normalen Sprachgebrauch die Handlung, die der Akteur ausgeführt hat. Hierbei kann jedes Verb gewählt werden, das gerade benötigt wird. Strenge Restriktionen soll es für die Wahl der Verben nicht geben.

Das Objekt des Statements stellt die Aktivität (Activity) dar, die der Akteur durchgeführt hat. Dabei können Activities auch frei definiert werden. Die Activity kann ein Link zu einer Webseite sein, ein WBT, eine Seite oder Lektion in einem WBT oder auch ein Kapitel in einem Buch, das der Akteur gerade liest.

[16] Siehe http://activitystrea.ms/ - Ein Projekt der Activity Streams Working Group in der mehrere Firmen (Microsoft, Google, IBM und viele mehr) zusammenarbeiten um eine Standardisierung der Activity Streams zu entwickeln.

Abbildung 20 - Erweitertes Statement inklusive Kontext und Ergebnis

Auch komplexere Statements sind möglich, indem man die Statements noch mit optionalen Komponenten, wie zum Beispiel einem Kontext oder einem Ergebnis (*Result*) erweitert.

➔ Durch die flexible Definition der Verben, Akteure und Objekte kann die Zielsetzung eines **flexiblen Daten-Modells**, mit dem man alle Arten von Aktivitäten beschreiben kann, erreicht werden.

➔ Durch die Tatsache, dass jedes Statement eine vollständige Beschreibung einer Aktion des Lernenden darstellt, ergeben sich neue Möglichkeiten für den Zugriff und die Verwaltung der Daten (Activity Streams in Lernerprofilen).

4.3.3.3 Die vier APIs der Experience API

Die Experience API Spezifikation beschreibt allerdings nicht nur eine API zur Übermittlung der Statements (Statement API). Zusätzlich werden noch drei sogenannte *Document APIs* definiert, die zusätzliche Aufgaben übernehmen sollen, allerdings auf der gleichen technischen Basis.

Statement API

Die Statement API bietet Funktionen, die soeben beschriebenen Statements in der „I did this“ Form und als JSON-Objekt formatiert an das LRS zu schicken, sowie Statements vom LRS anzufordern oder zu löschen.

State API

Die State API ist die erste der drei *Document APIs* der Experience API.

Mit ihr können *Activity Provider* benötigte Daten ablegen und zwischenspeichern, um sie geräteübergreifend und persistent verfügbar zu machen.

Dabei kann es sich zum Beispiel um den aktuellen Kurs-Status eines WBTs handeln. Ist der aktuelle Zustand in der State API gespeichert, kann das WBT diesen bei einem erneuten Aufruf wieder abrufen und sich in den letzten Zustand zurückversetzen.

Activity Profile API [17]

Die *Activity Profile API* ist die zweite *Document API* der Experience API.

[17] Es ist anzumerken, dass zum Zeitpunkt der Untersuchungen für diese Arbeit, die Arbeiten an der TLA noch nicht abgeschlossen waren und erst die Experience API als Konzeption und Umsetzung vorlag. Die Activity Profile API

Mit ihr können Datentupel vom Typ (Key, Value) im LRS gespeichert werden. Die Besonderheit ist, dass diese Daten in einem Profil einer einzelnen Aktivität eingebunden werden. Jede Aktivität, die mit der Experience API erfasst wurde, kann somit einen eigenen, sich dynamisch erweiternden Speicherort für Daten aller Art erhalten.

So könnte man mit der *Activity Profile API* zum Beispiel bei einem Quiz in einem WBT, die erreichte Punktzahl aller User, die die Frage bearbeitet haben, festhalten. Anderen Lernenden könnte dann durch Abruf und Integration der Daten in das Quiz, eine durchschnittlich erreichte Punktzahl der anderen Teilnehmer angezeigt werden.

Agent Profile API [17]

Die Agent Profile API ist die dritte und letzte *Document API* der Experience API.

Sie gleicht der Activity Profile API bis auf den Unterschied, dass die erfassten Datentupel nicht an eine Activity, sondern an einen Agent (z.B. die Userkennung eines WBT-Teilnehmers) gebunden werden können.

Somit könnte man Activity-übergreifend Daten an einen Benutzer binden. Dies könnte zum Beispiel dazu dienen, WBT-übergreifende Ergebnisse anzuzeigen oder Abhängigkeiten zwischen WBTs zu erstellen. So könnte WBT (B) durch Abruf von Daten aus der Agent Profile API erkennen, ob der aktuelle Benutzer WBT (A) schon ausgeführt, oder bestimmte Vorgaben schon erreicht hat.

4.3.4 Fazit Experience API

Das Modell der Experience API scheint die für die Umsetzung der API gesetzten Ziele zu erreichen und somit die TLA zu einer geeigneten Architektur für das aktuelle eLearning zu machen.

Die als WebService entworfenen und somit immer und von überall aus erreichbaren Learning Record Stores könnten es ermöglichen, einen freien und systemunabhängigen Zugriff auf die Daten anzubieten. Ebenso scheint die offen gestaltete Form der Statements es möglich zu machen, Aktivitäten ohne viele Einschränkungen zu beschreiben und abzuspeichern.

Das Konzept der Experience API scheint auf den ersten Blick gelungen, ob die technische Umsetzung der LRS als RESTful-WebService sowie der Statements als JSON-basierte Activity Streams allerdings vorbehaltlos geeignet ist, bleibt zu zeigen.

Sollte die Analyse zeigen, dass die technische Umsetzung der Experience API geeignet ist, so könnten damit die geforderten Ziele erreicht werden:

[17] weiter: und Agent Profile API sind in der Spezifikation der Experience API zwar definiert, aber Änderungen an ihnen sind noch möglich.
Der volle Nutzen und konkrete Anwendungsbeispiele für diese APIs sollten erst untersucht werden, wenn die TLA auch die noch fehlenden Komponenten (Learner Profiles, Competency Networks und Content Brokering) komplett umgesetzt hat. Für diese Komponenten wurden die beiden Profile APIs nämlich primär eingeführt.

➔ **Protokollierung von Erfahrungen jeder Art und Weise**
(Aktivitäten auf sozialen Plattformen oder auch Offline-Erfahrungen können erfasst werden)

➔ **System- und geräteunabhängige Erfassung von Aktivitäten**
(Die Aktivität muss nicht mehr in einem Browser stattfinden. Serious Games, Simulationen oder native mobile Apps werden möglich)

➔ **Verfügbarkeit der Daten**
(Die erfassten Daten können wieder abgerufen und von anderen Systemen weiterverwendet werden. Umsetzung von Learner Profiles oder externen Reporting Tools wird möglich)

4.4 Fazit – eLearning State of the Art

Durch die Untersuchung des *Horizon Reports* von 2012 und 2013 wurden aktuelle Anforderungen an das eLearning festgestellt und beschrieben. Diese Anforderungen bilden die Ziele, die jede aktuelle Entwicklung im Bereich eLearning zu erreichen versuchen sollte.

Mit der *Training and Learning Architecture* möchte die ADL aktuell einen neuen eLearning-Standard entwickeln. Welche Anforderungen beim Entwurf dieses Standards eine Rolle gespielt haben, und wie diese zustande kamen, wurde untersucht und diskutiert.

Vergleicht man die Anforderungen, die die TLA realisieren möchte, mit denen der aktuellen Trends aus dem Horizon Report, so kann man feststellen, dass sich diese gut ergänzen und die TLA somit ein geeignetes Konzept für die Umsetzung zeitgemäßer eLearning-Konzepte sein könnte.

Zentrale und zu diesem Zeitpunkt einzige umgesetzte Komponente der TLA ist die Experience API, die eine Umsetzung des Experience Trackings der TLA darstellt und die technische Basis der TLA bildet.

Die Konzeption der Experience API wurde vorgestellt und drei Hauptziele definiert, die eine technische Umsetzung der xAPI erreichen muss, um alle Anforderungen der TLA erfüllen zu können:

1. ***Geräte- und Systemunabhängigkeit***
2. ***Flexibles Daten-Modell***
3. ***Verfügbarkeit der Daten***

Das Modell, mit dem diese drei Eigenschaften erreicht werden sollen, wurde vorgestellt und für geeignet befunden, die von der TLA gestellten Anforderungen zu erfüllen. Ob allerdings die technische Umsetzung wirklich dazu in der Lage ist, die an sie gestellten Anforderungen zu erfüllen, bleibt zu untersuchen.

5 Analyse der Spezifikation der Experience API

Nachdem das Grundmodell der Experience API vorgestellt wurde, soll nun die technische Spezifikation untersucht werden. Dabei soll besonderes Augenmerk darauf gelegt werden, ob die im vorigen Kapitel definierten Ziele:

1. ***Geräte- und Systemunabhängigkeit***
2. ***Flexibles Daten-Modell***
3. ***Verfügbarkeit der Daten***

durch die technischen Spezifikation der Experience API erreicht werden konnten.

Als Quelle wird dabei die Spezifikation in Version 1.0 vom 26. April 2013 verwendet ([Ad13]). Da es sich hierbei um eine rein technische Spezifikation der Funktionen und über Regeln für die Verhaltensweise der einzelnen Funktionen handelt, wurden zum besseren Verständnis weitere Quellen hinzugezogen:

- [Ad12a] - Experience API Companion Document
- [Ru12c] - TinCan API - Statements 101
- [Ru12b] - TinCan API - Client QuickStart
- [Ru12a] - TinCan API - Client Library Guidelines

Einige dieser Quellen stammen noch aus dem Project TinCan und beschreiben demzufolge anstatt der Experience API die TinCan API. Da aber, wie in Kapitel 4.3.1 beschrieben, alle Angaben zur TinCan API auch auf die Experience API zutreffen, wird im Folgenden nur von der Experience API gesprochen werden.

5.1 Die technische Spezifikation der Experience API

Die technische Spezifikation der Experience API beinhaltet, neben einführenden Worten und Definitionen von Begriffen, zwei Hauptkapitel.

Als erstes wird das Datenmodell der Statements beschrieben und definiert. Dabei wird nicht nur auf die Struktur der Daten eingegangen, sondern es werden auch Regeln für das Verhalten des LRS bezüglich dieser Daten aufgestellt.

Darauf folgen die Definitionen der Datenübertragung (Data Transfer) zum LRS. Dies beinhaltet eine Beschreibung der vier APIs, die das LRS zur Verfügung stellen muss[1] und welche Antworten das LRS auf Anfragen senden muss.

In der Spezifikation wird somit zwischen Client (der die Daten in Form von Statements erzeugt und sendet) und Server (das LRS, welches die Daten empfängt und verfügbar machen kann) unterschieden und beide Seiten, teilweise gleichzeitig, definiert.

[1] Siehe Kapitel 4.3.3.3

Der Client stellt nach der Definition der Experience API Spezifikation einen Activity Provider dar. Also ein Programm, eine Webseite oder einen anderen Inhalt, der Daten an das LRS senden und von ihm anfordern kann.

Da die prototypische Umsetzung an einem WBT durchgeführt werden wird, soll in dieser Analyse ein WBT beispielhaft für einen Activity Provider stehen. Ein WBT, das die Experience API nutzt, wird im Folgenden als xAPI-WBT abgekürzt.

Zur Analyse der Experience API werden nun das Datenmodell der Statements sowie die Definitionen der vier APIs, die zur Datenübertragung genutzt werden, untersucht.

Dabei wird nicht die Reihenfolge aus der Spezifikation verwendet, sondern zuerst die APIs und anschließend das Datenmodell der Statements behandelt. Außerdem wird nicht auf alle Eigenschaften im Detail eingegangen, sondern nur auf die, die für diese Arbeit von besonderer Relevanz sind.

5.2 Data Transfer APIs

Wie bereits in Kapitel 4.3.3.3 beschrieben, besteht die Experience API nicht aus der Definition einer einzigen API, sondern insgesamt aus vier APIs: Der Statement API sowie drei Document APIs.

Alle APIs haben gemein, dass sie auf RESTful WebServices basieren.

„These four sub-APIs of the Experience API are handled via RESTful HTTP methods“[2]

Auf die allgemeine Technik eines RESTful WebService wird in der Spezifikation dabei nicht weiter eingegangen. Demzufolge muss man, um die Spezifikationen des LRS analysieren zu können, die allgemeinen Eigenschaften eines RESTful WebService untersuchen.

5.2.1 RESTful WebServices

Representational State Transfer (häufig auch RESTful WebService, RESTful API oder einfach nur REST-Server genannt) stellt eine Architektur zum Anfordern und Senden von Daten über das http-Protokoll dar, das aktuell sehr häufig zur Umsetzung von Web-Applikationen oder allgemein webbasierten Diensten verwendet wird.[3]

Ein RESTful WebService verwendet grundlegende vom http-Protokoll bereitgestellte Funktionen, um mit minimalem Overhead und somit sehr guter Performance Daten zu übertragen.

„REST ist einfach, aber klar definiert [...]“[4]

[2] Zitat: [Ad13]
[3] Mehr Informationen zu RESTful WebServices in [RR07] und [Mi12]
[4] Zitate aus der deutschsprachigen Veröffentlichung von [RR07]

> *„Wenn Sie REST-konforme Web Services bereitstellen, können Sie die bei der Komplexität eingesparte Zeit dafür einsetzen, zusätzliche Funktionalität anzubieten oder mehrere Services zusammenarbeiten zu lassen. [...] Je enger Sie sich an den grundlegenden Web-Protokollen orientieren, desto einfacher ist das."* [4]

Die Haupteigenschaften eines REST-Servers sind, dass dieser ressourcenbasiert und zustandslos arbeitet.

5.2.1.1 Ressourcenbasiert

Alle Daten, die auf dem REST-Server zur Verfügung gestellt werden, werden als Ressource bezeichnet. Der Zugriff auf diese erfolgt über eine repräsentative Darstellung der Ressource als *Uniform Resource Identifier* (URI).

Die eindeutige URI bildet dabei die Repräsentation der Ressource und setzt sich aus der http-Adresse des REST-Servers (`http://rest-server.com`) sowie der Ressource (`/username/123`), die angesprochen werden soll, zusammen.

Zum Beispiel: `http://rest-server.com/username/123`

Jede eindeutige URI beschreibt eine Ressource, die mit Hilfe verschiedener http-Request-Typen angesprochen werden kann.

Jeder Server, der eine Internetseite bereitstellt, ist in Grundzügen ein REST-WebService. Die Inhalte einer URL, die man in einem Browser angezeigt bekommen möchte, werden vom Browser durch einen http-GET-Request angefordert, empfangen und anschließend dargestellt. Somit sind die URLs die Ressourcenbeschreibung und die Webseiten die Daten, die der Server bei einem entsprechendem http-GET-Request übermittelt. Das Konzept der RESTful WebServer ist somit schon lange im Einsatz und kann von nahezu allen aktuellen Geräten, die einen Web-Browser bereitstellen, verwendet werden.

Die Besonderheit eines RESTful WebServices ist die Verwendung von weiteren Verben zur Verarbeitung von Daten. Neben http-GET-Requests an eine URI können ebenso POST- oder PUT-Requests zum Senden von Daten an den Server oder zum Beispiel DELETE-Requests zum Löschen von Daten verwendet werden.

So würde zum Beispiel: **GET** `http://rest-server.com/username/123`

den Usernamen vom User mit der ID 123 vom WebServer anfordern. Der WebServer würde die entsprechende Funktion ausführen, die dem http-GET-Request und der URI `/username/#Number` zugeordnet ist (sofern diese vorhanden ist) und den Usernamen des Nutzers an den Client im http-Request-Body zurückgeben.

DELETE `http://rest-server.com/username/123` fordert den WebServer zum Löschen dieser Ressource auf und

POST `http://rest-server.com/username/123` (mit den übergebenen Daten „Patrick") fordert den WebServer dazu auf, den Usernamen „Patrick" als User ID 123 zu setzen.

5.2.1.2 Zustandslosigkeit

Die Zustandslosigkeit des REST-Service bedeutet, dass jede Nachricht die an einen REST-Server geschickt wird eindeutig sein muss und alle Informationen beinhaltet, die der REST-Server zur Ausführung der Anfrage benötigt. Es werden keinerlei Zustandsinformationen zwischen zwei Nachrichten gespeichert. Nur so ist die Bedingung erfüllt, dass eine Anfrage an eine Ressourcen-URI immer das gleiche Ergebnis liefert und nicht zwischen verschiedenen Zuständen unterscheidet.

5.2.1.3 Datenformat

Für das Format der übertragenen Daten wird meist das JSON-Format[5] verwendet, da dieses einen sehr geringen Overhead bei der Darstellung von Daten besitzt. Auch andere Formate wie XML, HTML oder sogar binäre Daten sind möglich.

➔ Ein REST-WebService ist ein äußerst einfaches, aber effektives sowie zuverlässiges Modell, um Daten anzufordern und zu senden.

Außerdem ist es, durch die Verwendung von JSON und den einfachen http-Requests auf fast jedem System implementierbar. Die häufigste Anwendung finden RESTful WebServices in Verbindung mit JavaScript-Applikationen und insbesondere AJAX.

5.2.1.4 RESTful WebServices & JavaScript

Durch Verwendung von Ajax und speziell dem *XMLHttpRequest-Objekts*[6] ist es in JavaScript sehr einfach http-Requests an einen REST-WebService abzusetzen und zu verarbeiten. Werden die Daten dann noch im JSON-Format ausgetauscht, so können diese durch die native Unterstützung direkt von JavaScript verarbeitet werden. Somit sind Daten im JSON-Format für die Anwendung in JavaScript besser geeignet als zum Beispiel im XML-Format.

5.2.1.5 Cross-Origin Ressource Sharing

Ein Problem, das bei der Verwendung des XMLHttpRequest-Objekts beachtet werden muss, ist die *same-origin policy* Sicherheitsregel von JavaScript. Diese verbietet es dem XMLHttpRequest-Objekt Anfragen an andere Domains zu senden, als der, auf der sich der Nutzer gerade befindet. Dies soll das Nachladen von Schadcode oder die Übermittlung vertraulicher Informationen verhindern.

Eine Möglichkeit, diese Sicherheitsregel außer Kraft zu setzen, ist die Verwendung des *Cross-Origin Resource Sharing*[7]. Diese Funktion muss auf dem Server sowie auf dem

[5] Definition des JSON-Formats in [Cr06]

[6] http://www.w3.org/TR/XMLHttpRequest/ - Standardisierungs-Beschreibung für das XMLHttpRequest-Object des World Wide Web Consortium (W3C)

[7] http://www.w3.org/TR/cors/ - Cross-Origin Resource Sharing - W3C Candidate Recommendation

Client aktiviert und konfiguriert werden, damit domainübergreifende Zugriffe möglich werden.

Abbildung 21 – Unterstützung von CORS in Browsern

Unterstützt wird CORS von nahezu allen modernen Browsern, darunter auch die der mobilen Geräte, sodass die *same-origin policy* keine Einschränkung in der Verfügbarkeit der Daten darstellt, sofern man CORS verwendet.

5.2.1.6 Fazit - RESTful WebServices

Ruft man sich nochmals die Ziele in Erinnerung, die die technische Spezifikation der Experience API erreichen soll, so kann man feststellen, dass die Umsetzung des LRS als RESTful WebService das Erreichen dieser Ziele möglich macht und sogar eine äußerst performante Lösung darstellt.

Geräte- und Systemunabhängigkeit ✓

Http-Requests können von fast jedem Gerät und unabhängig davon, in welcher Programmiersprache der Inhalt entwickelt wurde, abgesetzt werden.

Flexibles Daten-Modell ✓

Das Format der übertragenen Daten von http-Requests an REST WebServices ist nicht vorgegeben. Üblich sind Daten im JSON-Format, jedoch auch XML, HTML oder sogar binäre Daten sind möglich.

Verfügbarkeit der Daten ✓

Wie jede normale Webseite, die über das http-Protokoll erreicht werden kann, ist ebenfalls der RESTful WebService ansprechbar. Durch die Verwendung von CORS entstehen auch mit JavaScript keine größeren Probleme dabei.

➔ Eine Umsetzung des LRS als RESTful WebService erfüllt alle an die Experience API gestellten Anforderungen. Zugriff auf das LRS und die darin gespeicherten Daten ist system- und geräteunabhängig möglich. Außerdem wird die Form der Daten, bis auf die Bedingung der Zustandslosigkeit der Daten, in keiner Weise eingeschränkt.

5.2.2 Die vier APIs der Experience API

Nachdem das allgemeine Konzept eines RESTful WebService beschrieben und zur Umsetzung der Experience API als geeignet befunden wurde, soll nun auf die Definition der vier APIs aus der Experience API Spezifikation eingegangen werden.

5.2.2.1 Statement API

Die Statement API ist die zentrale API der Experience API. Spricht man im Allgemeinen von der Experience API, so wird meist diese API und der allgemeine Umgang mit den Statements gemeint. Wie die Statements genau definiert sind, wird im nächsten Abschnitt behandelt. Zur Definition der Statement API genügt es davon auszugehen, dass es sich bei den Statements um Daten im JSON-Format handelt die, in der Form „I did this“, Aktivitäten beschreiben sollen.

Die Statement API stellt die benötigten Methoden bereit um Statements abspeichern sowie abrufen zu können. Die Ressource, welche den Zugriff auf die Statement API beschreibt, wird dabei unter `/statements` zur Verfügung gestellt.

Angenommen das LRS wäre unter `http://rest-server.com/lrs` erreichbar, so wären die http-Requests, die die Statement API akzeptieren muss, folgende:

```
PUT  http://rest-server.com/lrs/statements/statementID
POST http://rest-server.com/lrs/statements
GET  http://rest-server.com/lrs/statements/statementID
GET  http://rest-server.com/lrs/statements
```

PUT /statements/statementID speichert das übergebene Statement im LRS. Der Activity Provider (das WBT) generiert dabei die Statement-ID selbst. Das Statement muss, wie in der Beschreibung eines RESTful WebService vorgegeben, zustandslos sein. Das bedeutet, dass alle Informationen über Autor, Authentifizierung und alle Informationen über die getätigte Aktivität im Statement vorhanden sein müssen.

POST /statements speichert ein oder mehrere Statements gleichzeitig im LRS. Dabei muss der Activity Provider die Statement-ID nicht selbst generieren, sondern das LRS erzeugt sie selbständig und meldet in der Request-Response die ID des eingetragenen Statements an den Activity Provider zurück. Überträgt man mehrere Statements gleichzeitig in einem JSON-Objekt, so muss trotzdem jedes Statement vollständig definiert und zustandslos sein.

GET /statements/statementID fordert das Statement mit der ID vom LRS an.

Der Fall, dass der Activity Provider die Statement-ID kennt, ist allerdings nicht sehr häufig, da es meist keinen Sinn macht, sich die IDs der übertragenen Statements zu merken. Stattdessen ist es sinnvoller mit

GET /statements und einem JSON-Objekt als Übergabe, das die Suchkriterien definiert, zu arbeiten. Übermittelt man zum Beispiel die Daten eines Nutzers, in der wie in Kapitel 5.3.1 definierten Form, so bekommt man von der Statements API alle Statements, die

zu diesem Nutzer im LRS gespeichert sind. Dabei kann die Statement API die Anzahl der übergebenen Statements beschränken. Sind nicht alle Statements übergeben worden, so muss eine more-Ressource in den Rückgabedaten angegeben werden, über die die nächsten Daten abgerufen werden können.

Mit Hilfe der http-Request-Methoden können somit Statements vom Activity Provider gesendet sowie abgerufen werden. Wie der RESTful WebServer diese Funktionen umsetzen soll, wird nicht definiert, sodass der Server mit **beliebigen Techniken** entwickelt werden kann. Wichtig ist nur, dass die von ihm akzeptierten sowie ausgegebenen Daten der Experience API Spezifikation entsprechen.

Eine **Validierung** der Daten ist nur für die Syntax der Daten vorgesehen. So soll das LRS prüfen, ob alle benötigten Daten in einem Statement vorhanden sind und ob diese auch den Definitionen entsprechen. Ob die Daten Sinn ergeben oder korrekt sind soll (und kann), das LRS nicht entscheiden.

Die **Autorisierung**, ob ein bestimmter http-Request ausgeführt werden darf, wird vom LRS vor der Ausführung untersucht. Dafür können Statements die Eigenschaften *authority* tragen, das den *Agent* der den Request stellt, beschreibt. Aufgrund dessen kann das LRS dann entscheiden, ob der Request durchgeführt oder abgewiesen wird (siehe 5.3.1.10 - *Authority*).

➔ Mit Hilfe der Statement API können Statements von jedem Activity Provider gesendet sowie angefordert werden.

5.2.2.2 Document APIs

Die drei, in der Experience API Spezifikation beschriebenen Document APIs, dienen als zusätzliche Speicherorte für Daten der Activity Provider, Akteure oder Aktivitäten.

So basieren alle drei APIs auf den gleichen Regeln und unterscheiden sich im Vergleich zu der Statement API nur in der Definition der Ressourcenbeschreibung und der Art der zu übermittelnden Daten.

Die Ressourcen der drei Document APIs sind:

API	Ressource	Beispiel
State API	activities/state	http://rest.com/lrs/activities/state
Activity Profile API	activities/profile	http://rest.com/lrs/activities/profile
Agent Profile API	agent/profile	http://rest.com/lrs/agent/profile

Auf alle drei Ressourcen kann dabei mit PUT, POST, GET und DELETE zugegriffen werden.

Der Hauptunterschied der drei Document APIs zur Statement API ist, dass nicht nur der Content-Type „application/json“ verwendet werden kann, sondern jeder beliebige.

So können zum Beispiel auch binäre Daten an die Agent Profile API gesendet werde (ein Avatar-Bild für den User zum Beispiel). Die Beschreibung der Daten geschieht dabei als Datentupel [Key, Value] und es existieren keinerlei Restriktionen bezüglich der Namensgebung der Daten.

Wie bei der Statement API auch, kann man direkt bekannte *Activities* oder *Agents* durch Angabe der ID verwalten sowie durch Angabe von Informationen über das gesuchte Objekt danach suchen und sich eine Liste der Elemente vom LRS zurückgeben lassen.

Mit Hilfe der Activity State und Activity Profile APIs kann man Daten an eine bestimmte Aktivität und mit der Agent Profile API an ein User-Objekt binden.

Beispiele zur Verwendung sind:

- Der komplette Lerninhalt eines WBTs (Kurs) wird als Activity definiert. Der aktuelle Zustand des WBTs wird mit Hilfe der State-API in einem beliebigen Format abgespeichert (bevorzugt JSON) und kann somit beim Wiederausführen des Kurses wiederhergestellt werden. Dies kann sogar geräteübergreifend stattfinden, da der aktuelle Status im LRS und somit online verfügbar ist.
- Eine Frage, die als Activity definiert wurde, kann Daten, die sie betreffen in ihrem Activity Profile speichern (zum Beispiel eine Durchschnittspunkzahl oder die Anzahl der Abgaben).
- Ein Agent Profile kann als Lerner-Profil verwendet werden. So können erreichte Punkte, abgeschlossene Kurse oder auch Daten über den Lernenden (Avatar-Bild, Alter, Geschlecht, usw.) darin gespeichert werden.

➔ Mit Hilfe der Document APIs kann man erweiterte Informationen zu einzelnen Activities und Agents bereitstellen, die teilweise kursübergreifend verwendet werden können.

5.2.2.3 Fazit der Untersuchung der vier APIs

Betrachtet man abschließend die durch die Spezifikation definierten vier APIs unter den Gesichtspunkten der an die Experience API gestellten Anforderungen so kann man Folgendes feststellen:

Geräte- und Systemunabhängigkeit ✓
Verfügbarkeit der Daten ✓

Da alle vier APIs als RESTful APIs definiert wurden, gelten die gleichen Aussagen wie in Kapitel 5.2.1. Http-Requests können von fast wie jedem Gerät und unabhängig davon, in welcher Programmiersprache der Inhalt entwickelt wurde an die vier APIs gesendet werden. Somit können die Daten des LRS überall verfügbar gemacht werden.

Flexibles Daten-Modell ✓

Die Formate der Daten, die an die APIs übertragen werden können, sind bis auf die Statement API nicht beschränkt. In der Statement API vorgeschrieben, sowie bei den anderen APIs präferiert, ist jedoch das JSON-Format für Daten, was allerdings kein

großes Problem darstellt, da das JSON-Format von den meisten Entwicklungsmodellen unterstützt wird.

5.3 Statements

Nachdem die Definitionen der vier APIs der Experience API Spezifikation besprochen wurden wird nun die Definition der Form der Statements untersucht.

Statements sind Daten im JSON-Format, die an die Statement API gesendet sowie von ihr abgerufen werden können. Somit folgen die Statements der empfohlenen Form von Daten, die mit einem RESTful WebService ausgetauscht werden können.

5.3.1 Bestandteile der Statements

Statements werden im JSON-Format beschrieben.[8] Ein **JSON-Objekt** beinhaltet eine unsortierte Menge von Datentupeln im [Key, Value]-Format. Der Key gibt den Namen der Eigenschaft an und der Wert kann ein String, eine Zahl, ein Array oder wieder ein JSON-Objekt sein.

Ein Statement muss mindestens die Eigenschaften *actor*, *verb*, *object* und eine einzigartige ID haben. Somit folgt es dem „I did this“ Modell, das schon in Kapitel 4.3.3.2 vorgestellt wurde.

```
{
   "id": "12345678-1234-5678-1234-567812345678",
   "actor":{
      "mbox": "mailto:name@test.com"
   },
   "verb":{
      "id": "http://adlnet.gov/expapi/verbs/experienced",
      "display":{
         "en-US": "created"
      }
   },
   "object":{
      "id": "http://example.adlnet.gov/xapi/example/activity"
   }
}
```

Code 3 - Beispiel eines minimalen Statements

Demzufolge werden als erstes die drei Hauptbestandteile eines Statements vorgestellt und anschließend weitere, die zur Erweiterung der Statements genutzt werden können.

5.3.1.1 Actor

Ein *Actor* wird in der xAPI Spezifikation als einzelner *Agent* oder eine Gruppe von *Agents* angegeben.

[8] Definition des JSON-Formats in [Cr06]

Agent

Ein Agent stellt eine einzelne Person oder ein System dar, das Einfluss auf den Inhalt eines Statements hat. Ein Agent wird eindeutig durch einen der vier möglichen *Inverse Functional Identifiers (IFI)* definiert. Diese sind:

IFI	Beschreibung
mbox	Email-Adresse
mbox_sha1sum	Hash der Email-Adresse
openid	openID[9] des Nutzers
account	ein Useraccount auf einem externen System (zum Beispiel LMS) *wird als Objekt angegeben, das name des Users und homePage des Systems enthält.*

Ein Agent, der aufgrund seines Accounts auf einem anderen System identifiziert wird, würde also durch ein Objekt dieser Form definiert werden:

```
{
   "objectType": "Agent",
   "account": {
      "homePage": "http://www.example.com",
      "name":     "1625378"
   }
}
```

Code 4 - Beispiel eines Agents der durch einen Account definiert wird

Group

Ein Actor kann ebenso als Gruppe von Agents angegeben werden. In dem Fall ist der `objectType = „Group“` und ein Array der einzelnen Agents wird in der Eigenschaft `member` angegeben. So sind auch Teams oder Lerngruppen beschreibbar.

5.3.1.2 Verb

Das Verb in einem Statement beschreibt die Art der Aktivität, die der Actor an dem Objekt durchgeführt hat.

Dabei muss das Verb als *Internationalized Resource Identifier* (IRI)[10] angegeben sein. Diese IRI *kann* auch eine aufrufbare URL darstellen und einen Inhalt bereitstellen, der das angegebene Verb erklärt.

```
http://adlnet.gov/expapi/verbs/answered
http://adlnet.gov/expapi/verbs/attempted
http://adlnet.gov/expapi/verbs/completed
http://adlnet.gov/expapi/verbs/experienced
http://adlnet.gov/expapi/verbs/scored
```

Code 4 - Beispiele für Verben

Wie im Sprachgebrauch auch können gleiche Verben verschiedene Tätigkeiten in unterschiedlichen Kontexten beschreiben (besonders im Englischen). So kann „lost“ zum

[9] http://openid.net/ - Ein Webangebot einer einzelnen ID für mehrere Webseiten gleichzeitig

[10] Internationalized Resource Identifier – eine Form der Uniform Resource Identifier (URI). Somit sehr ähnlich einer URL. Ein IRI sollte eindeutig die Ressource beschreiben, muss allerdings nicht wie eine URL wirklich aufrufbar sein sondern stellt nur eine Beschreibung dar.

Beispiel das Verlieren eines Gegenstandes oder aber eines Spiels bedeuten. Ebenso könnte „ran“ bedeuten, dass der *Actor* gerade einen Marathon rennt oder ein bestimmtes Programm ausführt.

Ebenfalls kann in einem erstellen WBT zum Beispiel „completed“ heißen, dass alle Seiten gesehen wurden. In einem anderen WBT könnte es sein, dass der Kurs erst vollständig bearbeitet wurde, wenn alle Fragen beantwortet wurden oder wenn die volle Punktzahl erreicht wurde.

Dieser Doppelbedeutung soll entgegengewirkt werden, indem die Verben entweder gut gewählt oder eben mit einer aufzurufenden Webseite versehen werden, die eine genaue Definition des Verbes beinhaltet. So könnte jeder Hersteller eines WBTs eigene Webseiten anbieten, auf denen „seine“ Verben erklärt werden.

Dass dies zu Verwirrung führen kann, wurde auch von der ADL und der Community erkannt und so ist eine offene Datenbank für Verben geplant.

Allgemein gibt es allerdings keine Restriktion bezüglich der Verben (außer, dass diese in IRI-Form dargestellt werden müssen).

Zusätzlich zu dem Verb, kann noch die Beschreibung, sowie Übersetzung in verschiedenen Sprachen angegeben werden (siehe Kapitel 5.3.1.5 - Metadaten). So könnte ein Reporting-Tool das mit den gespeicherten Statements arbeitet, die Beschreibung auf der passenden Sprache anzeigen, sofern diese angegeben wurde.

```
{
   "verb" : {
      "id": "http://adlnet.gov/expapi/verbs/answered"
      "display":{
         "en-US": "answered",
         "de-DE“: "beantwortete"
      }
   }
}
```

Code 5 - Beispiel des Verbs "answered"

5.3.1.3 Object

Das Objekt des Statements ist das Element, auf das sich die Aktion des Nutzers bezieht. Das Objekt kann dabei eine Aktivität (Activity), ein anderer Agent oder gar ein anderes komplettes Statement sein.

Object = Activity

Ist das Objekt eine Activity, so wird diese durch den speziellen objectType „Activity“, einer ID und der Definition der Activity beschrieben.

Dabei muss die ID, wie die des Verbs, als IRI definiert sein. Auch hier ist es optional, ob diese wirklich aufrufbar ist und im besten Fall direkt auf die Aktivität verlinkt, oder ob sie nur beschreibend ist. Hauptanforderung ist, dass sie die Activity eindeutig definiert.

Eine Activity stellt somit ein spezielles Unterobjekt des *Objects* dar und wird durch folgende Eigenschaften beschrieben:

Eigenschaft	Beschreibung
name	Für Menschen lesbarer Name der Aktivität
description	Eine Beschreibung der Aktivität
type	Der Typ der Aktivität (z.B. course, page, question)
moreInfo	Weitere Informationen zur Aktivität. Eventuell mit Link der direkt zur Aktivität führt
Extensions	Beliebige weitere Daten, die zur Aktivität gehören
Interaction Properties **zB:** **interactionType** **correctResponsesPattern** **choices**	Weitere Eigenschaften, die an die in SCORM 2004 verwendeten cmi.interaction – Objekte angelehnt sind. *Dies entspricht einer Umsetzung des AICC/CMI-Standards und soll Aktivitäten auch durch automatisierte Prozesse auswertbar machen.*

Eine Activity kann somit ein gesamtes WBT darstellen sowie eine einzelne Frage aus diesem. Ebenso können auch Aktivitäten aus dem Alltag damit abgebildet werden, indem die Beschreibung passend gewählt wird.

```
{
   "id": "http://lrs.com/Kurs3/Seite5/MatheAufgabe1",
   "definition": {
      "name": {
         "de-DE": "Mathematik-Aufgabe 1",
         "en-US": "Math-Quesition 1"
      },
      "description": {
         "de-DE": "Ist 1+2=3 ?",
         "en-US": "Is 1+1=3 ?"
      },
      "type": " http://lrs.com/activities/cmi.interaction"
      "interactionType": "true-false",
      "correctResponsesPattern": [ "true" ]
   },
   "objectType": "Activity"
}
```

Code 6 - Beispiel eines Objects vom Typ Activity

Die Angabe der Interaction Properties machen es Reporting-Tools möglich, die Aktivitäten automatisch zu bewerten, wie Autoren es von SCORM bisher gewöhnt sind. Dabei ist die Angabe allerdings optional und eine Aktivität kann auch einfach nur durch einen Namen definiert sein.

Object = Agent

Das Objekt des Statements kann sich ebenso auf einen anderen Agent beziehen. Somit könnte man zum Beispiel ausdrücken, wenn sich zwei Nutzer vernetzt haben. Auch das bekannte „like“ von Facebook wäre dadurch repräsentierbar.

```
{
   "id": "12345678-1234-5678-1234-567812345678",
   "actor":{
      "objectType": "Agent",
      "account": {
         "homePage": "http://www.facebook.com",
         "name": "Patrick"
         }
      },
   "verb":{
      "id":"http://facebook.com/likes",
      "display":{
         "en-US": "likes"
         "de-DE": "mag"
      }
   },
   "object":{
      "objectType": "Agent",
      "account": {
         "homePage": "http://www.facebook.com",
         "name": "Katja"
   }  }
}
```

Code 7 - Beispiel eines Facebook-Likes in Statement Form

Object = Ein anderes Statement

Komplexere Aussagen können getätigt werden, wenn man als Objekt des Statements die Referenz-ID eines anderen Statements angibt. So könnten Aktivitäten bewertet, kommentiert oder allgemein markiert werden.

```
{
   "id": "12345678-1234-5678-1234-567812315678",
   "actor" : {
      "objectType": "Agent",
      "mbox":       "mailto:tutor@harvard.com"
   },
   "verb": {
      "id": "http://harvard.com/tutor/rated",
      "display": {
         "en-US": "rated"
      }
   },
   "object" : {
      "objectType": "StatementRef",
      "id":         "8f87cade-bb56-4c2e-ab83-44982ef22df0"
   },
   "result" : {
      "rating": "5 stars for this great answer!"
   }
}
```

Code 8 - Beispiel eines Statements das sich auf ein anderes Statement bezieht

Alternativ kann man sogenannte Sub-Statements verwenden. Also anstatt der ID des anderen Statements, das komplette Statement als Objekt angeben.

➔ Die Möglichkeiten Statements alleine mit den drei Grundbausteinen Actor, Verb und Objekt zu bilden sind schon sehr groß und den mit SCORM erfassbaren Daten überlegen. Fast jeder Satz der mit „I did this“ beschrieben werden kann, sollte durch die sehr offene Definition der einzelnen Elemente der Statements beschreibbar sein.

Durch die Einführung weiterer, ergänzender Eigenschaften werden die Möglichkeiten, die man mit den Statements hat, sogar noch größer.

5.3.1.4 Result

Zusätzlich zu Actor, Object und Verb kann man noch das Ergebnis dieser Aktion beschreiben (wie in Code 8 schon verwendet). Dafür kann man verschiedene Eigenschaften des *Results* setzen. Diese sind: *score, success, completion, response, duration.*

5.3.1.5 Metadaten

Fast alle Eigenschaften eines Statements können mit Metadaten beschrieben werden.

Dabei können mindestens die Metadaten *name* und *description* gesetzt werden. Bei *Activities* können zusätzlich noch *type* und *moreInfo* belegt werden[11]. Zusätzlich können alle Metadaten, mit Hilfe der *RFC 5646 Language Tags*[12], in unterschiedlichen Sprachen angegeben werden. [13]

Alternativ können Metadaten zu einer Eigenschaft (zum Beispiel einer Activity) auch von einer URL abgerufen werden, sofern deren ID eine URL darstellt.

Diese Art der Angabe von Metadaten ist erst in Version 1.0 der Spezifikation hinzugefügt worden. Bis Version 0.95 beschrieb die Spezifikation der Experience API eine Datei im XML-Format, die wie schon bei SCORM, die Metadaten des xAPI-WBT beinhalten sollte und im Hauptverzeichnis des xAPI-WBT vorhanden sein musste. In ihr sollten die Metadaten über die einzelnen Aktivitäten und den Kurs selbst bereitgestellt werden sowie das Content-Packaging für LMS beschrieben werden. (siehe auch Kapitel 5.4)

Die *tincan.xml* ist jedoch nicht mehr Teil der Experience API Spezifikation sondern wurde in Version 1.0 entfernt und die Beschreibung der Metadaten durch ein auf JSON-basiertes System umgestellt.

So soll das LRS, sollte es auf eine Activity treffen, dessen ID eine URL ist, versuchen diese URL zu laden und aus dem abgerufenen Inhalt die Metadaten extrahieren.

Metadaten sollen also nicht mehr in einer zentralen Datei (wie bei SCORM) gesammelt werden, sondern können sich direkt auf der Webseite der Activity befinden. Sollte die Activity kein Web-Inhalt sein, kann als Activity-ID trotzdem eine URL angegeben werden unter der nur die Metadaten und die Beschreibungen dieser Activity zu finden sind.

Im Beispiel eines Serious Games könnte dies folgendermaßen umgesetzt werden: Angenommen der Spieler muss mehrere Level durchlaufen, dann könnte jedes Level eine Activity darstellen. Unter der URL `http://game.com/Level1` würden die Entwickler

[11] Siehe 5.3.1.3 - Objects

[12] Standard zur Identifizierung von Sprachen: http://tools.ietf.org/html/rfc5646
Zum Beispiel: „de-DE“ für Deutsch, „en-US“ für amerikanisches Englisch

[13] Wie in den vorangegangen Beispielen zu sehen.

eine Webseite bereitstellen, die die Metadaten sowie Beschreibungen für diese Activity im JSON-Format für das LRS bereitstellt. Diese wird vom LRS abgerufen und damit die Activity eindeutig beschrieben.

Diese Lösung bietet eine gute Möglichkeit die Metadaten, die zu einzelnen Activities gehören, separiert vom Hauptinhalt zu pflegen. So kann im Serious Games Beispiel die Beschreibung losgelöst vom Content aktualisiert oder verändert werden. Die Aktualisierung und Pflege der *tincan.xml* fällt weg. Der Autor muss nach dem Einfügen einer Seite in ein WBT nicht die Einträge in der *tincan.xml* ergänzen und damit jede Activity nochmals einzeln aufzählen, sondern liefert die Metadaten einfach direkt auf der einzelnen Seite des WBT mit aus.

Demzufolge funktioniert diese Lösung gut für Inhalte, die auf HTML-Basis erstellt werden und im Internet distribuiert werden, da diese die Metadaten direkt mitliefern können. Für native Applikationen bedeutet dies allerdings den Mehraufwand der Erstellung der Internetseite, auf der die Metadaten vorhanden sein müssen. Beide Applikationsformen können allerdings auch auf diese Art der Metadaten-Angabe verzichten und die Metadaten einfach in jedem Statement mitsenden (siehe oben).

Somit ergeben sich folgende Spezifikationen für Metadaten:

- Metadaten können für fast alle Eigenschaften von Statements angegeben werden.
- Die Metadaten können mehrsprachig sein.
- Die Metadaten können in den Statements mitübertragen werden.
- Metadaten können, sofern die ID der Eigenschaft eine URL ist, vom LRS direkt von der angegebenen URL abgerufen werden.

5.3.1.6 Context

Mit Angabe einer *Context*-Eigenschaft, kann man das Statement noch in einen Kontext mit verschiedenen anderen Dingen setzen. Zum Beispiel:

- in dem Kontext eines anderen Statements.
- im Kontext einer Parent- oder einer Gruppe von Activities, auf die sich das Statement bezieht oder von denen es Teil ist.
- im Kontext zu einem Agent (zum Beispiel dem Lehrer oder einer Arbeitsgruppe).

Somit können Verknüpfungen und Beziehungen verschiedener Statements hergestellt werden.

Jede bearbeitete Frage oder Seite eines WBTs kann zum Beispiel im Kontext des WBTs angegeben werden. Somit könnten Hierarchien und Abhängigkeiten innerhalb der Lerninhalte abgebildet werden.

5.3.1.7 TimeStamp

Jedes Statement sollte einen Timestamp in sich tragen, sodass die zeitliche Abfolge der Statements jederzeit wieder rekonstruiert werden kann. Auch die Darstellung als Activity Stream wird dadurch realisierbar.

5.3.1.8 Attachments

Auch Anhänge sind in einfacher Form möglich. So kann als Anhang an ein Statement eine durch eine URL erreichbare Webseite angegeben werden, unter der ein Zertifikat oder Ähnliches bereitgestellt wird. Ebenso ist es möglich, direkt binäre Daten in das Statement einzufügen. So könnte man Bilder oder andere Dateien miteinander austauschen.

5.3.1.9 Extensions

Zusätzlich, zu den schon aufgezählten Eigenschaften der Statements, kann Statement-, Activity- sowie Result-Objekten noch die *Extensions*-Eigenschaft zugeordnet werden.

In diesen Extension-Objekten können JSON-Daten sowie binäre Daten abgelegt werden, die zu den jeweiligen übergeordneten Objekten gehören und erweiterte Informationen dazu beinhalten.

Extension-Objekte in einem Statement sollten Informationen über den Kontext bereitstellen. In einem *Result*-Objekt wären dies weitere Informationen, die zu diesem Ergebnis geführt haben und in Activities Informationen, die die Activity ausführlicher oder in einem speziellen Format, beschreiben.

➔ Durch die ergänzenden Eigenschaften Result, Context, TimeStamp, Attachments, Extensions sowie die Verwendung von mehrsprachigen Metadaten werden die Möglichkeiten, um selbst komplexe Aktivitäten durch Statements zu beschreiben, ausgeweitet und der Funktionsumfang der Statements erweitert.

Ein weiteres, bislang außenvorgelassenes Thema, ist die Sicherheit der Statements. Wie stellt das LRS sicher, dass nur Clients, die das Recht dazu haben, Zugriff auf die Daten bekommen? Die Sicherheit von Daten ist heutzutage ein wichtiges Thema und gerade bei Datenspeichern im Internet von enormer Bedeutung. Zu diesem Zweck wurde die Eigenschaft Authority eingeführt.

5.3.1.10 Authority

Die *Authority*-Eigenschaft eines Statements beinhaltet Informationen darüber, wer den http-Request, zum Senden oder Abfragen von Statements, an das LRS gesendet hat.

Dabei wird das Authority-Objekt als ein einzelner Agent oder eine Gruppe mit zwei Agents beschrieben. Bei den Agents kann es sich um Nutzer der Inhalte sowie die Activity Provider selbst handeln, die als Agents definiert werden.

Als Authentifizierungs-Methoden mit dem LRS werden die http-Basic-Authentication[14] (Abfrage von Usernamen und Passwort) sowie eine oAuth2.0-basierte Authentifizierung[15] beschrieben.

Weiterhin sollte ein LRS die Möglichkeit bieten, Applikationen in ihm zu registrieren und sie mit Username und Passwort auszustatten. Außerdem sollte es Einstellungen geben, die die Rechte einzelner Applikationen einschränken können. So könnte man WBTs, die größtenteils nur Daten schreiben, Zugriff auf fast alle Funktionen des LRS gestatten, aber ein Abrufen der *Agent Profiles* verbieten, sodass keine personenbezogenen Daten durch das WBT ausgelesen werden können.

Das Abrufen von kritischen Informationen könnten dann nur Agents durchführen, die die Autorisierung dazu haben. Das könnten entweder die Besitzer der *Agent Profiles* sein, die einen Account im LRS haben oder spezielle Reporting-Tools, die entsprechend abgesichert sind und die Daten anonymisieren oder die Erlaubnis zur Verwendung haben.

Auf dieser Basis unterscheidet die Spezifikation zwischen verschiedenen Kombinationen der Agents und der Authentifizierungs-Methode.

Zur Erklärung der folgenden Szenarien wird angenommen, dass es sich bei dem *Activity Provider* um ein WBT handelt, das einen Lerninhalt mit Aufgaben sowie einer End-Auswertung beinhaltet.

Authority-Objekt nicht vorhanden oder Agents haben keine Rechte

Wird das ein http-Request ohne jegliche Authority-Angaben an das LRS gesendet oder keiner der angegeben Agents hat das Recht zur Verwendung der angeforderten Funktion des LRS, so sollte das LRS eine http-Basic-Authentication fordern. Ein Fenster mit Username und Passwort sollte den Nutzer des WBTs dazu auffordern, sich zu authentifizieren. Kann er dies nicht tun, oder wird kein dem LRS bekannter Account mit den benötigten Rechten angegeben, wird der Zugriff auf die angeforderte Ressource verweigert.

Authority-Objekt ist eine bekannte Applikation + unbekannter User

Meist ist es aber der Fall, dass das WBT dem LRS bekannt ist. Also dass es einen Usernamen und Passwort zur Authentifizierung im LRS hat, mit dem es bestimmte Funktionen des LRS verwenden kann.

Angenommen also, das Authority Objekt ist eine Gruppe bestehend aus zwei Agents. Einer davon ist ein WBT und dem LRS als Applikation bekannt. Der andere Agent ist der Nutzer, der keinen Account im LRS hat und somit dem LRS unbekannt ist.

[14] http://tools.ietf.org/html/rfc2617 - HTTP Authentication: Basic and Digest Access Authentication
[15] http://oauth.net/2/ - OAuth 2.0 Homepage

In diesem Fall kommt es darauf an, welche Rechte die Applikation im LRS zugeteilt bekommen hat. Werden http-Requests an Ressourcen ausgeführt, auf die es nicht zugreifen darf, wird vom User eine http-Basic-Authentifizierung verlangt, um sich einzuloggen und somit zu überprüfen, ob der User das Recht dazu hat, die Ressource anzufordern.

In einem WBT könnte man dies dermaßen umsetzen: Sollte ein Student, der dem LRS als User nicht bekannt ist, das WBT bearbeitet, aber das WBT sich beim LRS authentifiziert hat, so darf das WBT Daten senden und allgemeine Daten zu den einzelnen Activities abrufen (Durchschnittspunkte oder Ähnliches).

Kommt der Student dann auf die Auswertungsseite sieht er nur seine eigenen Ergebnisse des Kurses, welche aus den im State gespeicherten Daten angezeigt werden. Zugriff auf die Agent Profile API hat er nicht, da er dem LRS nicht bekannt ist.

Authority-Objekt ist eine bekannte Applikation + bekannter User

Hat er allerdings einen Studenten-Account im LRS und ist somit dem LRS bekannt und eindeutig zu einem Agent Profile zugeordnet, so wird die Auswertung im WBT ihm die Daten aus dem Agent Profile anzeigen (eventuell noch mit Punkten aus anderen Kursen an denen er teilgenommen hat).

Für einen Autor, der einen Account im LRS hat und auch die entsprechenden Rechte besitzt, könnte der Aufruf der Auswertung andere Daten liefern. So könnte der WBT-Autor, sofern er es mit seinen Studenten geklärt hat, Zugriff auf alle Agent Profiles erhalten und alle Daten der Teilnehmer sehen.

Authority-Objekt ist eine unbekannte Applikation + bekannter User

Für den Fall, dass der User dem LRS bekannt ist, aber die Applikation nicht, soll ein oAuth-Verfahren eingeleitet werden. Das heißt, dem User wird angezeigt, dass eine unbekannte Applikation versucht, Zugriff auf seine Daten zu bekommen.

Stimmt er dem zu, so kann das WBT oder das Fremd-Tool mit einer temporären Authentifizierung auf das Agent-Profile des Agents zugreifen.

Dieser Fall wäre zum Beispiel gegeben, wenn der User auf einer Webseite ein eigenes Lerner-Profil mit der Experience API verwaltet. Angenommen dieses Tool bietet die Möglichkeit Agent Profiles von anderen LRS zu importieren, so könnte der User durch die oAuth-Authentifizierung im LRS die Webseite dazu berechtigen, auf seine Daten zuzugreifen.

➔ Die Experience API Spezifikationen behandeln das Thema Sicherheit kurz, aber effektiv. Durch die Anforderung an ein LRS, dass dieses oAuth2.0- und eine http-Basic-Authentication bereitstellen muss, können viele Szenarien abgedeckt werden die durch moderne Datensicherheitsanforderungen entstehen.

➔ Allerdings fordert dies eine nicht unkomplizierte Erweiterung für das LRS. So ist das LRS nicht mehr nur ein Datenspeicher auf REST-Basis, sondern muss auch eine User-, Applikation- und Rechteverwaltung integrieren, um alle Anforderungen der Datensicherheit erfüllen zu können.

5.3.2 Fazit zur Definition der Statements

Zusammenfassend kann man somit für die Definition der Statements feststellen, dass durch die sehr offene und freie Definition der Statements die Forderung nach einem flexiblen Daten-Modell eindeutig erfüllt wurde.

Flexibles Daten-Modell ✓

Alleine schon durch die Definition der Haupteigenschaften der Statements kann eine sehr hohe Flexibilität in der Erfassung von getätigten Aktivitäten erreicht werden. So genügen die drei Objekte Actor, Verb und Object, um fast jeden Satz, der in der einfachen Form „I did this“ gebildet werden kann, zu beschreiben und somit auch im LRS zu erfassen.

Die weiteren Definitionen der zusätzlichen Eigenschaften Result, Context, TimeStamp und Extensions steigern nochmals die Aussagekraft der erfassten Statements, sodass selbst komplexe Statements inklusive ihres Kontextes abbildbar sind. Durch die Definition von mehrsprachigen Metadaten sowie der Möglichkeit Attachments an die Statements anzuhängen, wird der Funktionsumfang sogar noch über das Ziel der einfachen Datenprotokollierung hinaus erweitert.

Zusätzlich muss man, durch die Möglichkeit den AICC/CMI5-Standard[16] zur optionalen Beschreibung von Aktivitäten verwenden zu können, nicht auf die von SCORM bekannten Funktionen zur automatischen Auswertung von Interactions verzichten.

Geräte- und Systemunabhängigkeit ✓

Die Verwendung von JSON als Format für die Statements stellt kein Problem dar, da JSON-Implementierungen für fast alle Entwicklungsumgebungen existieren. Im Gegenteil, JSON bietet den Vorteil ein sehr leichtgewichtiges und für den Menschen gut lesbares Format zu sein.

Verfügbarkeit der Daten ✓

Die Verfügbarkeit der Daten wird primär durch die Definition des LRS bestimmt, jedoch wird in den Statements das Sicherheitskonzept der Experience API mit dem Authority-Objekt definiert. Ist der Zugriff auf Daten nicht abgesichert, sollte man sie nicht verfügbar machen. Die Anforderung an eine Implementierung von oAuth2.0 und der http-Basic-Authentication schafft die nötigen Voraussetzungen, um die Verfügbarkeit der Daten abzusichern und vor Zugriff von Unbefugten zu schützen.

[16] Siehe 5.3.1.3 – Object

5.4 Offline-Phasen des Activity Providers

Da es sich um das LRS um einen WebService handelt, muss zur Kommunikation mit ihm eine Internetverbindung verfügbar sein.

Bei mobilen Geräten sowie Inhalten, die über einen längeren Zeitraum verwendet werden (Serious Games, lange Self Assessment WBTs oder Simulationen), kann es dazu kommen, dass temporär keine Verbindung zum Internet besteht. Auf diesen Punkt geht die Spezifikation der Experience API jedoch nicht ein.

In einem Protokoll eines Meetings der *Tin Can API Working Group* aus der Entwicklungsphase der Experience API (TinCan API) kann man jedoch herauslesen, dass dieses Problem durchaus bedacht, jedoch für die Aufnahme in die Spezifikation als nicht geeignet eingestuft wurde. So wollte man den Entwicklern der Activity Provider keinerlei Vorschriften zur Art der Speicherung von nicht-übersendbaren Statements machen.

> *„Nik: there is some traffic about queuing statements in regards to offline storage – local storage and offline files for web apps. In JavaScript this is easy, how do we do it in Native? We don't want to drive this for content. You can store it however you want. We can come up with solutions in a Best Practices Guide. The Tin Can Library Guidelines. [...]"* [Ti12]

Betrachtet man die Tin Can Library Guidelines [Ru12a], so kann man in ihr das Kapitel *Statement Queuing and Offline Storage* finden. Darin wird die mögliche Funktion von Activity Providern zur Zwischenspeicherung und Einreihung in eine Warteschlange (Queue) für Statements beschrieben.

So ist es für WBTs, die HTML5 und JavaScript zur Umsetzung benutzen, durch Verwendung des *LocalStorage*[17] des Browsers möglich, Statements, die wegen einer Offline-Phase des WBTs nicht gesendet werden konnten, in einer Queue abzuspeichern. Diese können dann, sollte eine Verbindung zum Internet wieder bestehen, nachträglich an das LRS übermittelt werden.

Durch die Eigenschaft der Vollständigkeit eines jeden Statements sowie der Statement-Eigenschaft TimeStamp, entstehen dabei keinerlei Probleme, sollte das LRS Statements zeitverzögert empfangen. Die Definition der Statement API, dass der http-POST-Request an `/statements` es auch erlaubt mehrere Statements gleichzeitig an das LRS zu übertragen, vereinfacht das Abarbeiten der Queue zusätzlich.

➔ Verhalten des Activity Providers während Offline-Phasen werden in der Spezifikation nicht behandelt. Statement-Queuing ist dafür eine elegante Lösung.

[17] Siehe [Pi10]

5.5 Content Packaging

Bis Version 0.95 beschrieb die xAPI-Spezifikation eine Methode zum Content Packaging von Inhalten mit Experience API-Unterstützung durch eine Datei im xml-Format, die, wie schon bei SCORM, das Content Packaging und die Metadaten des xAPI-WBT beinhalten sollte. In ihr sollten die einzelnen Aktivitäten des WBTs, Metadaten für die Aktivitäten sowie Launch-Informationen für das LMS beschrieben werden.

Seit Version 1.0 der xAPI-Spezifikation ist die tincan.xml nicht mehr Teil der Spezifikation, da diese verworfen wurde (siehe auch Kap 5.3.1.5 – Activity Metadaten).

Somit existieren momentan keine Regeln für das Content Packaging. Allerdings hat dies den Hintergrund, dass das Content Packaging auch sinngemäß nicht in die Spezifikation einer API passt. Vermutlich wird für das Content Packaging in späteren Phasen der TLA-Entwicklung eine separate Spezifikation entworfen, die den Import in LMS möglich macht.

xAPI-WBTs müssen zwar nicht mehr in einem LMS bereitgestellt werden, es sollte den Autoren aber die Möglichkeit gegeben werden, dies trotzdem zu tun, falls sie die Verwaltung ihrer Lerninhalte mit einem LMS bevorzugen, da sie keinen eigenen Server zur Distribution der WBTs unterhalten möchten.

Möglichkeit 1 – SCORM Content Packaging weiterverwenden:

Die Einbindung der Experience API in ein WBT kann zusätzlich zur Einbindung von SCORM geschehen. Somit hat der Autor die Möglichkeit, das SCORM Content Packaging (also den IMS-Standard) zu verwenden, um das WBT in ein LMS zu importieren.[18]

Der Vorteil dieser Methode ist, dass das WBT sich direkt nach dem Aufruf mit dem LMS über die SCORM Run-Time Environment verbinden kann, und somit die Benutzerdaten des LMS über den Lernenden erhält. Mit diesen Daten kann dann ein Actor-Account für die Experience API erstellt werden und die Kommunikation mit dem LRS kann eindeutig dem Lernenden des LMS zugeordnet werden.

Nachteil dieser Methode ist die notwendige zusätzliche Implementierung von SCORM. Das WBT muss zwar nur die Basisfunktionen von SCORM bereitstellen. Jedoch gelten dadurch auch alle Einschränkungen in der Distribution von SCORM-Paketen.

Möglichkeit 2 – Externe Verlinkung des WBT

Alternativ kann der Autor in einem LMS auch einen Link auf das WBT als externe Ressource setzen. Das WBT wird also auf einem separaten Server hochgeladen und nur im LMS verlinkt.

Nachteil dieser Variante ist jedoch, dass das WBT keinerlei Informationen über den Lernenden erhalten kann und somit keinen Actor erstellen kann. Die erfassten

[18] Siehe Kapitel *3.5.2.1 - Content Aggregation Model - CAM*

Statements wären somit anonym oder müssten durch eine vom WBT bereitgestellte zusätzliche Authentifizierung im LRS erfasst werden.

In Version 0.95 der Spezifikation wurde die Möglichkeit angegeben, Actor sowie andere Variablen in der Link-URL als Parameter zu übergeben. Diese Passagen wurden jedoch auch entfernt. Natürlich kann der Autor diese Funktionalität selbst nachimplementieren, jedoch wäre eine Lösung für diese Problematik in Form einer Spezifikation wünschenswert.

➔ Content Packaging Spezifikationen fehlen noch.

5.6 Fazit der Analyse der technischen Spezifikation der Experience API

Fasst man die Analyse der technischen Spezifikation der Experience API zusammen, so kann man feststellen, dass durch die sehr offene und freie Definition der Statements sowie der Umsetzung des LRS als RESTful WebService alle definierten Ziele der technischen Spezifikation der Experience API erfüllt werden konnten.

5.6.1 Erreichte Ziele der Spezifikation

Flexibles Daten-Modell ✓

Betrachtet man zuerst das **Format der Daten**, so kann man feststellen, dass alle drei Document-APIs so gut wie keine Beschränkung im Format der übertragbaren Daten definieren. Sogar binäre Daten können im [Key, Value]-Format übertragen und im LRS gespeichert werden. Alleine die Statements sind fest im JSON-Format definiert, wobei auch hier die frei wählbare Extensions-Eigenschaft Autoren die Möglichkeit bietet, beliebige Daten in Statements zu integrieren. Die Wahl des JSON-Formats stellt dabei keine Einschränkung für den Autor dar, da alle Daten die mit anderen Formaten, wie zum Beispiel XML, abbildbar sind, auch im JSON-Format umgesetzt werden können. Somit kann das Format der Daten als sehr flexibel bewertet werden.

Durch die, wie schon in 5.3.2 - *Fazit zur Definition der Statements* beschriebene, sehr lose Definition der Inhalte der einzelnen Eigenschaften der Statements sowie der Erweiterung durch viele Nebeneigenschaften, die zur Verfeinerung der Beschreibung einer Aktivität eingesetzt werden können, ist auch das Ziel, ein **flexibles Daten-Modell** zu schaffen, erfüllt worden.

Eine weitere Eigenschaft der Statements liegt in der Implementierung des LRS als RESTful WebService und der daraus resultierenden Eigenschaft der Zustandslosigkeit der Statements. Statements enthalten immer vollständige Beschreibungen ihrer einzelnen Eigenschaften, sodass diese dynamisch angelegt werden können, wenn sie das erste Mal an das LRS gesendet werden. Keine der Eigenschaften, die in einem Statement angegeben sind, muss dem LRS vorher bekannt sein. Die mühevolle

Definition von *Objectives* und *Interactions* in der *imsmanifest.xml* von SCORM[19] fällt somit weg.

Geräte- und Systemunabhängigkeit ✓

Die Definition des LRS als ein RESTful WebService führt zu der Möglichkeit, Anfragen an das LRS von so gut wie jedem System sowie von fast jedem Gerät, das sich mit dem Internet verbinden kann, zu senden. Die Verwendung von RESTful WebServices ist heutzutage weitverbreitet und es existiert eine Vielzahl von Frameworks, die Entwicklern bei der Implementierung eines RESTful WebServices und somit auch eines LRS unterstützen können.

Auch die Verwendung von JSON als Format für die Statements stellt kein Problem dar. JSON-Implementierungen sind für fast alle Entwicklungsumgebungen verfügbar und somit können die Statements von der Umsetzung im JSON-Format profitieren und die Vorteile eines sehr leichtgewichtigen und für den Menschen gut lesbaren Formats nutzen.

Verfügbarkeit der Daten ✓

Wie schon bei dem Ziel der Geräte- und Systemunabhängigkeit wird der Grundstein für eine hohe Verfügbarkeit der Daten nach außen durch die Definition des LRS als RESTful WebService gelegt. Die Daten können zum LRS gesendet sowie von ihm abgerufen werden, soweit die nötige Authentifizierung des Clients gegeben ist. Dabei spielt es keine Rolle, um was es sich bei dem Client handelt. Der Client, der die Daten vom LRS fordert, kann eine Webseite sein, ein WBT, eine native App für ein beliebiges Gerät oder gar ein anderes LRS, das Daten synchronisieren möchte. All dies spielt keine Rolle, solange die Anfrage korrekt gestellt ist und die Autorisierung einen Zugriff auf die Ressource erlaubt.

Die von RESTful WebServices geforderte Vollständigkeit der Daten und somit auch die Forderung zur Vollständigkeit der einzelnen Statements trägt dazu bei, dass keine Anfrage „sinnlose“ Ergebnisse liefert. Jedes Statement ist in sich geschlossen und bedarf keiner weiteren Informationen, um interpretierbar zu sein.

➔ Somit scheinen alle Hauptziele der technischen Spezifikation der Experience API erreicht worden zu sein. Der Autor bekommt ein höchst flexibles Daten-Modell zur Beschreibung von Aktivitäten, die Implementierung der Experience API sollte Geräte- und Systemunabhängig durchgeführt werden können und die Daten im LRS sind von überall erreichbar.

Betrachtet man auch die, durch die Analyse aktueller Trends im eLearning gestellten **Anforderungen an aktuelle Technologien für das eLearning**[20], so kann man auch unter

[19] Siehe *3.5.2.1 - SCORM - Content Aggregation Model*

[20] Wie in *4.1.4 - Anforderungen aktueller Trends an das eLearning* zusammengefasst

diesen Anforderungen das Modell der Experience API und somit auch der Training and Learning Architecture als gelungen bezeichnen.

> *„Aktivitäten von Lernenden sollten auch außerhalb eines LMS protokollierbar und bewertbar sein, sodass auch externe Tools wie soziale Plattformen in den Lernprozess integriert werden können.“* ✓
>
> *„Die erfassten Daten sollten aussagekräftiger werden, um auch LMS-Externe Aktivitäten gut beschreiben zu können. Ebenso sollten die Daten nicht nur im LMS vorhanden sein, sondern nach außen erreichbar gemacht werden, um sie besser wiederverwerten zu können.“* ✓
>
> *„Die Technik zur Protokollierung von Aktivitäten sollten Geräte- und auch Browserunabhängig verwendbar sein, sodass auch Out-Of-Browser Szenarien wie native Apps oder Serious Games möglich werden.“* ✓

Die Experience API ist somit eine Spezifikation für eine Technologie, die den Anforderungen des aktuellen eLearnings durchaus gerecht wird. Viele neue und aktuelle Trends können durch die Verwendung der Experience API realisiert werden. Zum Bespiel:

- Mobile WBTs ohne Einschränkungen der Entwicklungsmethode (WebApps, hybride Apps sowie native Apps) können erstellt werden.
- Daten aus Serious Games oder Simulationen können erfasst werden.
- Aktivitäten in LMS-externen Tools wie sozialen Plattformen können protokolliert werden.
- Learning Analytics könnte auf Basis der im LRS verfügbaren Daten durchgeführt werden.

Auch für die Training and Learning Architecture stellt sie eine geeignete Basis dar. Die TLA ist zwar in ihren weiteren Teilen noch im Entwicklungsprozess, die Experience API kann jedoch als technische Basis für sie als durchaus geeignet eingeschätzt werden.

5.6.2 Probleme der Spezifikation

Während der Analyse der aktuellen Spezifikation der Experience API konnten jedoch auch einige Probleme festgestellt werden, auf die auch kurz eingegangen werden soll.

So ist die die **technische Spezifikation** allgemein als technisches Dokument zur Implementierung der Experience API grundsätzlich gelungen. Es werden kleinere Beispiele mitgeliefert sowie LRS und Client zwar gleichzeitig aber verständlich beschrieben und die Funktionen, die von ihnen bereitgestellt werden müssen, klar definiert. Jedoch ist gerade das Kapitel über die **Sicherheit** der Statements zu kurz geraten. Ohne ein vorhandenes Wissen über die Funktionsweise von oAuth oder allgemeine Vorgehensweisen zur Authentifizierung sind die Definitionen hier zu komplex geraten, um leicht zugänglich zu sein und die Anforderungen eines „einfachen“ Standards zu erfüllen.

Es ist allerdings anzumerken, das Version 1.0 der technischen Spezifikation der Experience API nur 80 Seiten umfasst. Abzüglich Einleitung und Anhänge bleiben 55 Seiten für die komplette Definition aller Funktionen inklusive kleiner Beispiele. Dies ist nicht viel im Vergleich zu den über 1000 Seiten der SCORM 2004 Spezifikation. Somit ist die Spezifikation durchaus kompakt und für die Fülle an Informationen, die sie beinhaltet, gut aufgebaut. Mit geschätzten 20-30 Seiten mehr, gefüllt mit Hintergrundinformationen und ergänzenden Beispielen, könnte die Spezifikation allerding besser verständlich und der Zugang für einen technisch unversierten Leser leichter werden.

Hierzu sei erwähnt, dass so wie das komplette Project TinCan, auch die Entwicklung der Spezifikations-Dokumente der Experience API, als Community-Projekt betrieben wird. So kann man unter https://github.com/adlnet/xAPI-Spec die offene Entwicklung der Experience API verfolgen und daran teilnehmen, wenn man möchte. Es können ebenso „Bugs“ oder Ergänzungen gepostet werden, die dann von der Community korrigiert und eingepflegt werden. Es ist also anzunehmen, dass die Dokumentationen der Experience API mit der Zeit immer besser, ausführlicher und auf spezielle Anwendungsfälle bezogen werden.

➔ Technische Spezifikation ist teilweise zu kurz.

Ob die als flexibel und offen bewertete **Definition der Daten** wirklich allen Anforderungen genügt, werden Zeit und die wachsende Zahl an Anwendungsfällen zeigen. So könnte es sein, dass die freie Definition von Verben und Activities auch zu vielen Missverständnissen führt, da keine festen Vorgaben für diese existieren. Allerdings ist sich die ADL dieses Problems bewusst und spricht dieses in der Spezifikation der Experience API selbst an. Als Lösungsansatz wird weiterhin auf eine gute Zusammenarbeit mit der Community gesetzt und die Einführung einer offenen Plattform zum Verwalten von Verben könnte hierbei die Lösung für das Problem der Fragmentierung von IDs für Verben und Activities sein.

➔ Fragmentierung von Verben und Activities möglich.

Das besprochene Fehlen von Definitionen für das **Content Packaging** in der Spezifikation ist nicht befriedigend.[21] Es kann verstanden werden, dass für Content Packaging sinngemäß eine API-Dokumentation nicht der richtige Platz ist, an dem es definiert werden sollte. Allerdings wurde es versäumt, eine entsprechende, alleinstehende Spezifikation für das Content Packaging zu entwerfen. Natürlich ist eines der großen Features der Experience API, dass WBTs nicht mehr an LMS gebunden sein müssen, sondern überall distribuiert werden können. Jedoch sollte man den Autoren wenigstens die Möglichkeit bieten, ihren bekannten Work-Flow mit der Nutzung eines LMS beizubehalten, ohne auf alle Features der Experience API verzichten zu müssen.

[21] wie in Kapitel 5.4 behandelt

Es ist davon auszugehen, dass, sollte die Experience API eine breite Verwendung finden, die LMS sich an die Experience API anpassen werden und entsprechende Lösungen wie offenen Webspace für den Autor oder Repository Systeme zur Verfügung stellen werden, in denen die WBTs distribuiert werden können. Im Moment ist das fehlende Content Packaging jedoch ein Stolperstein, der nicht nötig gewesen wäre.

➔ Content Packaging fehlt noch

Eine Vorgehensweise, wie reagiert werden sollte, wenn das LRS wegen fehlender Verbindung zum Internet in **Offline-Phasen** nicht erreichbar ist, gibt es in der Spezifikation auch nicht. Eine Statement Queue, wie in 5.4 beschrieben, ist aber wohl die einfachste Lösung für das Problem und sollte auf fast jedem System umsetzbar sein.

➔ Regeln für Offline-Phasen des Activity Providers existieren nicht.

Wie jedoch schon erwähnt, ist es sehr wahrscheinlich, dass diese kleineren Mängel im Laufe der Zeit und mit Fortschreiten des Projekts TLA sowie der Experience API behoben werden. Das grundsätzliche Konzept der Experience API ist gelungen und die aufgezählten Mängel leicht korrigierbar. Sie sollten als nicht allzu schwerwiegend eingeschätzt werden.

Abschließend zur Analyse der Experience API Spezifikation soll noch die Erfüllung des allgemein gefassten, großen Ziels der ADL

„Make it simple…
Make it powerful…“ [Ad12b]

kritisch betrachtet werden. Die Experience API schafft es, den Autoren eine sehr große Freiheit beim Umgang mit Daten von Lernenden zu ermöglichen. Dabei ist die Form sowie der Zugriff auf die Daten so flexibel gestaltet, dass er kaum Wünsche offen lässt und allgemein auf Höhe aktueller Technologien ist. Das Ziel ein „mächtiges“ Modell zu definieren, erfüllt die Experience API somit. Das Ziel „simpel“ zu sein kann allerdings nicht vollständig erreicht werden. So ist es für Autoren im Vergleich zu SCORM nicht unbedingt einfacher, die Statements zu generieren und abzusenden. Die Möglichkeit, viel mehr Informationen in den Statements abbilden zu können als im SCORM Daten-Modell führt dazu, dass man die Daten, die die Statements benötigen, erst einmal zur Verfügung haben muss. Der Aufwand ein „komplettes“ Experience API-WBT zu erstellen ist somit höher als die Erstellung eines SCORM-kompatiblen WBTs. Natürlich bekommt man dafür auch den Mehrwert der ausdrucksstärkeren Daten, aber eben zu dem Preis, dass die Anwendungen komplexer und datenintensiver entwickelt werden müssen.

Die richtige Aussage wäre also eher *„Make it powerful and as simple as possible“.* Also mächtig und dabei so einfach wie möglich. Dieses Ziel kann mit der Experience API momentan erreicht werden.

➔ Die Experience API ist nicht einfach für den Autor umzusetzen.
Aber so einfach wie möglich, angesichts der vielen Möglichkeiten, die sie bietet.

Ob eine Implementierung der Experience API mit aktuellen technischen Mitteln überhaupt möglich ist und wie schwierig oder einfach diese Implementierung umzusetzen ist, soll im nächsten Schritt an einer prototypischen Integration in ein WBT sowie der Erstellung eines prototypischen LRS untersucht werden. Dabei werden nur die wichtigsten Funktionen implementiert, um sich ein Bild über die nötigen Arbeitsschritte sowie des benötigten Aufwandes machen zu können.

Außerdem soll untersucht werden, ob die Spezifikation der Experience API während der Implementierung vielleicht doch noch Probleme aufwirft, die während der technischen Analyse nicht sofort erkennbar geworden sind.

6 Implementierung der Experience API

Nachdem im vorangegangen Kapitel die technische Spezifikation der Experience API analysiert und als geeignet zur Umsetzung eines, den aktuellen Anforderungen des eLearning entsprechenden WBTs eingestuft wurde, soll nun die Implementierung der Experience API in ein bestehendes WBT sowie die prototypische Umsetzung eines LRS vorgenommen werden.

Diese Implementierungen sollen dazu dienen, um zu überprüfen, ob alle von der Spezifikation der Experience API geforderten Methoden und Komponenten mit den aktuellen technischen Möglichkeiten umsetzbar sind, oder ob dabei Probleme entstehen, die in der rein theoretischen Analyse der Spezifikation übersehen wurden oder nicht abzusehen waren.

Als Basis der Implementierung wird dabei ein Prototyp eines mit dem LernBar Autorensystem erstellten, auf HTML5 und JavaScript basierenden, WBTs verwendet. Somit wird zuerst das LernBar Autorensystem kurz vorgestellt und das zur Implementierung verwendete WBT beschrieben.

Anschließend wird die Experience API sowohl auf Client- (WBT) als auch Serverseite (LRS) prototypisch durchgeführt. Dabei werden die Auswahl der verwendeten Techniken sowie die Vorgehensweise bei der Umsetzung beschrieben.

Abschließend sollen Auffälligkeiten und Probleme, die während der Implementierung aufgetreten sind, besprochen werden. Außerdem wird untersucht, ob die Implementierung mit den gewählten technischen Mitteln, im Hinblick auf die Anforderungen der Spezifikation der Experience API, erfolgreich durchgeführt werden konnte.

6.1 Das LernBar Autorensystem – Basis der prototypischen Umsetzung

Das LernBar Autorensystem wird seit 2005 von studiumdigitale[1] (ehemals megadigitale), der zentralen eLearning-Einrichtung der Goethe-Universität Frankfurt, entwickelt. Dabei stellt das LernBar Autorensystem ein Template-basiertes Autorensystem für WBTs dar, das sich am besten durch ein Zitat der Homepage des LernBar Autorensystems[2] beschreiben lässt:

> *„Das Autorensystem LernBar ermöglicht eine einfache Umsetzung didaktischer Strukturen, ohne dass sich Autoren um den gestalterischen Aufbau des Bildschirms oder die Navigation kümmern müssen. Über 50 Gestaltungsvorlagen unterstützen*

[1] http://www.studiumdigitale.uni-frankfurt.de/ - studiumdigitale Webseite
[2] http://www.studiumdigitale.uni-frankfurt.de/et/LernBar/index.html - LernBar Autorensystem

> *Einsteiger als auch Fortgeschrittene beim Anlegen von Inhaltsseiten, Übungen und Testaufgaben. Zahlreiche Fragetypen und Feedbackmöglichkeiten sowie das Einbinden aller gebräuchlichen Medienformate ermöglichen die schnelle Umsetzung der unterschiedlichsten Lernszenarien."*

Die aktuellste Version des LernBar Autorensystems ist *LernBar Release 3w* (im Folgenden R3w genannt), wobei zusätzlich eine Preview der Version *Release 4* (im Folgenden R4 genannt) bereits auf der Homepage heruntergeladen werden kann.

Im LernBar Autorensystem enthalten ist das LernBar Studio. Eine Desktop-Applikation für Windows-PCs, mit der Autoren WBTs auf Template-Basis erstellen können. Dabei kann der Autor einzelne Seiten mit verschiedenen Fragetypen versehen sowie den Kurs in Lektionen einteilen und Navigations- oder Zeitbeschränkungen für die Seiten, Lektionen oder den gesamten Kurs erstellen.

Die erstellten Kurse können anschließend als SCORM 1.2 oder SCORM 2004 kompatibles Paket exportiert werden und somit in jedem SCORM-kompatiblen LMS distribuiert werden. Alternativ kann der Autor sein erstelltes WBT im LernBar Portal hochladen. Dabei handelt es sich um ein Web-Portal, das Autoren eine einfache Rechteverwaltung zur Regelung des Zugriffs auf die hochgeladenen WBTs ermöglicht. Weiterhin bietet es für Studenten die Möglichkeit, sich einen Account anzulegen und die für sie bereitgestellten WBTs auszuführen.[3]

Bis einschließlich Version R3w konzentrierte sich das LernBar Autorensystem auf die Erstellung von WBTs für herkömmliche Desktop-PCs. Die verwendeten Technologien dabei sind HTML, JavaScript sowie Adobe Flash[4], das zur Umsetzung von interaktiven Elementen (Medien, Fragetypen, Auswertungsseiten) der erstellten WBTs verwendet wird.

Version R4 soll auf die aktuellen Anforderungen des eLearnings eingehen und den Nutzern zusätzlich die Möglichkeit bieten, mit dem LernBar Autorensystem erstellte WBTs (im Folgenden LernBar-WBTs genannt) ebenso auf mobilen Geräten ausführen zu können. Dies war bislang, aufgrund der mangelnden Verbreitung von Adobe Flash auf mobilen Geräten wie Tablets oder Smartphones, nicht möglich.

Dabei werden für den Autor keinerlei zusätzliche Schritte während der Erstellung des Kurses benötigt. Die Desktop- sowie die mobile Variante der LernBar-WBTs greifen beide auf die gleichen Daten zu, um den Kurs wie gewünscht darzustellen. Die mobile Variante verzichtet dabei jedoch auf Flash und setzt den LernBar Player, der in der Desktop-Variante die Hauptkomponente des WBTs darstellt, auf JavaScript-Basis um. Der LernBar-Player stellt in der Desktop-Variante Funktionen wie die Navigation durch den Kurs, die Kommunikation mit SCORM sowie fast alle anderen wichtigen Funktionen

[3] Weitere Informationen zum LernBar Autorensystem finden sich auf den angegebenen Webseiten sowie in diesem, zu Informationszwecken von studiumdigitale bereitgestelltem, WBT: http://lernbar.uni-frankfurt.de/mdigi/LBE3

[4] http://www.adobe.com/de/products/flashruntimes.html - Informationen zu Adobe Flash

bereit. Ebenso werden alle Fragetypen und Medien, die in der Desktop-Variante mit Flash realisiert wurden, durch den Einsatz von HTML5 und JavaScript neu umgesetzt.

Abbildung 22 – Eine Seite aus LernBar Release 3 (oben) sowie die gleiche Seite in der mobilen Version aus Release 4 (links)

LernBar Release 4 befindet sich im Moment noch im Entwicklungsstadium. Die Neuentwicklung aller Komponenten ist noch nicht abgeschlossen, allerdings sind die wichtigsten Funktionalitäten wie das Navigieren durch die einzelnen Seiten des WBTs sowie einfache Fragetypen wie Multiple- und Single-Choice Aufgaben schon umgesetzt.

Das WBT, in dem die Implementierung der Experience API stattfinden soll, stammt direkt aus der Entwicklungsphase des LernBar Autorensystems und ist noch nicht final. Die mobilen LernBar-WBTs werden mit HTML5, JavaScript sowie den Bibliotheken jQuery und jQuery Mobile umgesetzt. Bei mobilen LernBar-WBTs handelt es sich somit um WebApps, die der Nutzer auf einem mobilen Gerät im Browser ausführen kann. [5]

Da ein großer Teil von WebApps, klassischen WBTs und Webseiten im Allgemeinen auf Basis von HTML5 und JavaScript entwickelt werden, soll die prototypische Implementierung der Experience API in das LernBar-WBT exemplarisch für all diese Inhalte vorgenommen werden. Wie schon in *4.1.3 - mLearning und mobile WBTs* besprochen, sollten viele der gewonnenen Erkenntnisse während der Implementierung ebenso auf hybride Apps anwendbar sein, da diese größtenteils dieselben Techniken zur Umsetzung einsetzen.

Auf weitere Besonderheiten des, mit dem LernBar Autorensystem erstellten, WBTs wird im weiteren Verlauf eingegangen, sofern sie von Relevanz während der Implementierung sind.

[5] Mehr Informationen zur Entwicklung der mobilen Umsetzung von LernBar-WBTs sowie der verwendeten Technik finden sich in [Be12].

6.2 Integration der Experience API in ein WBT

Bevor die Implementierung der Experience API durchgeführt werden kann, muss zuerst eine geeignete Position im vorhandenen Programmablauf für die Umsetzung gefunden werden. Dafür wird zuerst der grobe Ablauf der Initialisierung eines LernBar-WBTs besprochen.

Anschließend werden einzelne Techniken, die zur Implementierung der Experience API verwendet werden können, besprochen und die Wahl der Techniken zur Umsetzung der Implementierung der Experience API begründet.

Abschließend wird die Implementierung der Experience API vorgestellt, wobei beispielhaft einige der erstellten Methoden im Detail betrachtet werden.

6.2.1 Bestandteile und Ablauf eines LernBar-WBTs

Um eine, für die Implementierung der Experience API, geeignete Position im Programmcode zu finden, müssen zuerst die Bestandteile und der Ablauf eines LernBar-WBTs untersucht werden.

Ein mit dem LernBar Autorensystem Release 4 erstelltes WBT wird dem Autoren im ZIP-Format ausgegeben. In dieser ZIP-Datei sind alle für den Aufruf des WBTs benötigten Dateien enthalten.

Im Hauptverzeichnis eines LernBar-WBTs befindet sich eine index.html, die die Startdatei des WBTs darstellt. In dieser wird über eine Browserweiche[6], die den *UserAgent* des Browser ausliest, entschieden, ob es sich um ein mobiles oder ein herkömmliches Desktop-Gerät handelt, welches das WBT aufruft. Je nach Gerät wird anschließend auf die entsprechende Version weitergeleitet.

Da die Implementierung in der mobilen Version (ohne Adobe Flash) geschehen soll, wird somit nur die Weiterleitung auf die mobile.html oder in der vorliegenden Entwicklungsversion auf mobile-dev.html, betrachtet.

In der mobile.html werden alle benötigten JavaScript-Dateien eingebunden und geladen. Ebenso enthält sie alle HTML-Elemente, die für die Darstellung des WBTs benötigt werden.

Alle JavaScript-Dateien, die für die Umsetzung des WBTs entwickelt und verwendet werden, befinden sich im Ordner `\lernbar\js`. Alle JavaScript-Objekte sind im Namespace `STDT.Lernbar` modularisiert und gekapselt, um alle Objekte und Methoden des LernBar-WBTs sinnvoll zu strukturieren und verfügbar zu machen.[7] Die

[6] http://detectmobilebrowsers.com/ - Die DetectMobile-Browserweiche

[7] Weitere Informationen zu Modularisierung und Kapselung von JavaScript-Objekten sowie JavaScript im Allgemeinen in [Cr08] und [Ma13].

Ordnerstruktur des Ordners `lernbar/js/src` bildet dabei die verfügbaren Module ab und macht somit einen Zugriff auf die einzelnen Dateien leichter.

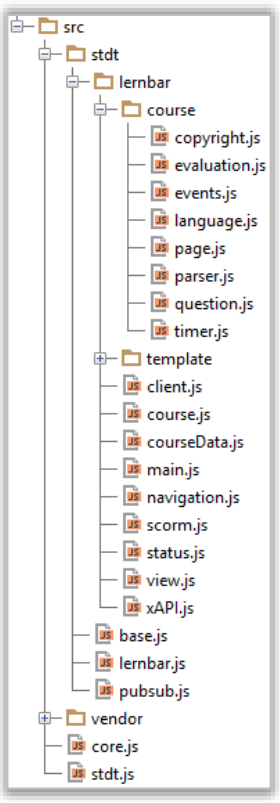


Abbildung 23 – Module eines Lernbar-WBTs

Weiterhin existiert im Programmcode eine strikte Trennung zwischen den Modulen zur Bereitstellung funktionaler Komponenten des WBTs (Navigation im Kurs, Verarbeitung von Fragen) sowie den Modulen zur Darstellung des WBTs.

Somit kann bei der Implementierung der Experience API auf eine Analyse der Module, die zur Darstellung des Kurses verwendet werden, verzichtet werden. In diesen sind keine Anpassungen notwendig. Nur in die Komponenten die den Kursablauf steuern soll eingegriffen werden um die benötigten Methoden für die Nutzung der Experience API zu erstellen.

Im Folgenden wird somit der grobe Programm-Ablauf beschrieben, um einen Überblick über die zur Implementierung wichtigen Module zu erhalten.

1. **`STDT.Lernbar.main.initialize()`** - Die Methode, die zur Initialisierung eines LernBar-WBTs gestartet wird, befindet sich im Modul `STDT.Lernbar.main`. Es beinhaltet alle, zur Initialisierung des Kurses benötigte, Funktions-Aufrufe.

2. **`course.initialize(course.xml)`** – Die `course.xml` aus dem Hauptverzeichnis sowie alle anderen benötigten Daten des Kurses werden in dieser Methode geladen und mit Hilfe eines Parsers in ein JavaScript-Objekt umgewandelt, um alle Informationen über das WBT zur Verfügung zu haben.[8]

3. **`STDT.Lernbar.courseData`** wird erzeugt – Hierbei handelt es sich um ein global erreichbares Objekt, das im weiteren Programmablauf allen weiteren Modulen alle Daten über den Kurs zur Verfügung stellt.

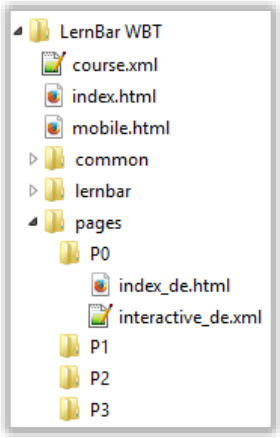


Abbildung 24 - Dateien eines LernBar-WBTs

4. **`STDT.Lernbar.client.initialize()`** – Die Initialisierung des Clients, der alle Funktionen des WBTs steuert, wird durchgeführt. In diesem Modul sollen die Experience API sowie SCORM-Funktionalitäten implementiert werden.

[8] Bei der course.xml handelt es sich um eine Datei im XML-Format, in der alle Informationen über das WBT enthalten sind. So befindet sich in der course.xml die Struktur des Kurses, im Besonderen die Struktur aller verfügbaren Lektionen inklusive der darin enthaltenen Seiten und alle damit verknüpften Regeln (Navigations- und Zeitbeschränkungen).
Zusätzlich existiert für jede einzelne Seite des WBTs ein Unterordner im pages-Ordner des WBTs. In jedem pages-Ordner befindet sich eine weitere html-Datei mit dem Namen index_de.html in der der Inhalt der einzelnen Unterseite gespeichert ist. Enthält die Seite Fragen oder Medien, so beinhaltet der Ordner zusätzlich noch eine interactive_de.xml, die die jeweiligen Einstellungen der Frage oder des Videos beinhaltet (siehe Abbildung 24).

5. `STDT.Lernbar.view.render()` - Nach der Initialisierung werden die Daten des WBTs an den Renderer des WBTs weitergereicht. Dieser stellt das WBT mit Hilfe von jQuery Mobile sowie des Frameworks Handlebars[9] dar.

Somit wird die Implementierung der Experience API hauptsächlich im Modul `STDT.Lernbar.client` durchgeführt. In dieser werden alle Hauptfunktionen des WBTs initialisiert und somit stellt sie auch einen geeigneten Ort zur Initialisierung der Experience API Methoden dar.

6.2.2 Verwendete Techniken zur Implementierung der Experience API

Zur Implementierung der Experience API in ein bestehendes, auf HTML5- und JavaScript-basierendem WBT bietet sich die Nutzung verschiedener aktueller Techniken an.

Betrachtet man die Anforderungen, die die Experience API an einen Activity Provider (das WBT) stellt, so muss dieser zwei Aufgaben erfüllen.

1. Die Statements müssen in Form von **JSON-Objekten** generiert werden.
2. Der Activity Provider muss durch **http-Requests** mit dem LRS kommunizieren.

Die **Generierung der Statements** benötigt dabei keine weiteren Techniken. Da das WBT schon JavaScript verwendet, müssen die benötigten Daten nur aus den schon vorhandenen Modulen der WBT-Implementierung extrahiert werden und als JavaScript-Objekt in Form eines Statements zusammengefasst werden (siehe 5.3).

Das Senden der ***http-Requests*** kann allerdings durch die Verwendung verschiedener Techniken geschehen.

So können http-Requests an das LRS einerseits direkt durch Verwendung des von JavaScript bereitgestellten XMLHttpRequest-Objekts konfiguriert und abgesendet werden[10].

Alternativ kann man Frameworks wie *jQuery* verwenden, die verschiedene Methoden bereitstellen, um die Browser-Kompatibilität von verschiedenen JavaScript Methoden zu erhöhen und somit ihre Verwendung erleichtert. Gerade in Hinblick auf die Verwendung von *CORS* zur Übermittlung der Daten (siehe 5.2.1.5), bietet die Verwendung, der von jQuery bereitgestellten Methode *jQuery.ajax()*[11], massive Erleichterungen im Unterschied zu einer direkten Verwendung des XMLHttpRequest-Objekts.

Mit Hilfe von jQuery kann man somit relativ einfach die benötigten Ressourcen des LRS ansprechen und die Statements vom WBT übertragen.

[9] http://handlebarsjs.com/ - Handlebars Template Framework

[10] http://www.w3.org/TR/XMLHttpRequest/ - Standardisierungs-Beschreibung für das XMLHttpRequest-Object des World Wide Web Consortium (W3C)

[11] http://api.jquery.com/jQuery.ajax/ - die Methode Ajax() von der jQuery-Bibliothek

```
jQuery.support.cors = true;          //CORS aktivieren
jQuery.ajaxSetup({cache: false});    //Caching deaktivieren
jQuery.ajax({
    async: false,
    url: LRSUrl,                     // die URL des LRS
    type: "GET",                     // oder POST, DELETE
    data: {},                        // das Statement-Objekt
    success: function (response)  { /* Bei Erfolg auszuführen */},
    error: function (errorThrown) { /* Bei Fehler auszuführen */}
});
```

Code 9 - Beispiel für einen jQuery.ajax() Aufruf

TinCanJS – Bibliothek

Speziell an die Bedürfnisse der Implementierung der TinCan API (und somit auch der Experience API) angepasste Bibliotheken stellt Rustici Software in ihrem GitHub https://github.com/RusticiSoftware/ bereit. Vorhanden sind Bibliotheken für Objective-C, Java sowie auch für JavaScript die TinCanJS genannt wurde[12]. Alle Bibliotheken werden unter der *Apache 2.0 License* vertrieben, was eine private sowie kommerzielle Nutzung möglich macht.

Die TinCanJS-Bibliothek ist noch nicht in einer finalen Version erschienen, allerdings bietet sie bereits die grundlegenden Methoden, um Statements generieren, verwalten, verifizieren und senden zu können. Dabei unterstützt TinCanJS den Entwickler dabei, die Statements entsprechend der in der Experience API Spezifikation aufgestellten Regeln zu definieren und stellt Methoden zur Kommunikation mit dem LRS bereit. TinCanJS verwendet dafür nicht jQuery.ajax(), sondern direkt das XMLHttpRequest-Objekt von JavaScript. Allerdings werden auch hier Ausnahmen und Regeln zur Steigerung der Browser-Kompatibilität sowie die Verwendung von CORS integriert.

Ein weiterer Vorteil von TinCanJS ist, dass es, ebenso wie die Experience API, öffentlich entwickelt wird. So kann jeder an der Entwicklung von TinCanJS mitarbeiten. Momentan wird TinCanJS regelmäßig aktualisiert und erweitert.

Für die Implementierung der Experience API wird somit die Bibliothek TinCanJS verwendet, um die Experience API möglichst regelkonform umzusetzen.

SCORM_API_Wrapper.js – Bibliothek

Zusätzlich zur Implementierung der Experience API werden SCORM-Grundfunktionalitäten in das WBT integriert, um eine Distribution des WBTs in LMS möglich zu machen (siehe 5.5 - Content Packaging).

Dafür wird der *JavaScript SCORM API Wrapper* von Philip Hutchison verwendet, der auf https://github.com/pipwerks/scorm-api-wrapper heruntergeladen werden kann und unter einer MIT-style Lizenz vertrieben wird.

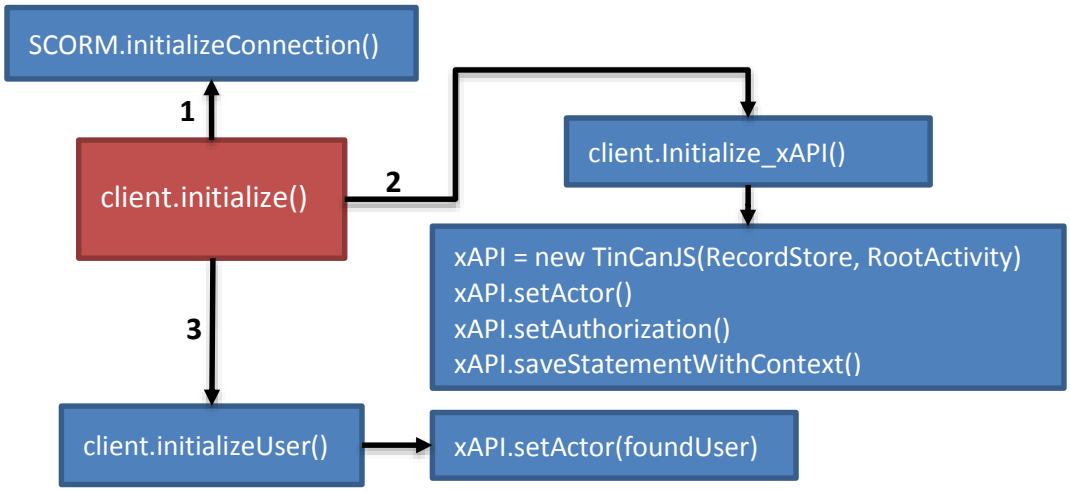
[12] https://github.com/RusticiSoftware/TinCanJS – GitHub der TinCanJS

Die JavaScript Bibliothek SCORM_API_Wrapper stellt dabei Methoden zur einfachen Verbindung mit der SCORM Run-Time Environment sowie zur Übermittlung und zum Empfang von Daten aus der SCORM API bereit.

6.2.3 Implementierung der Experience API

Nachdem in 6.2.1 das Modul `STDT.Lernbar.client` des bestehenden WBT-Programmcodes als Einstiegsposition der Initialisierung der Experience API festgelegt und die Verwendung der TinCanJS als zusätzliche Bibliothek zur Umsetzung gewählt wurde, soll die Implementierung nun vorgestellt werden.

Sind, wie in 6.2.1 beschrieben, alle Daten des Kurses initialisiert worden, wird die Methode `STDT.Lernbar.client.initialize()` aufgerufen. In dieser Methode werden verschiedene Objekte initialisiert, um anschließend allen Modulen zur Verfügung zu stehen.



Code 10 - Darstellung der Initialisierung der TinCanJS

6.2.3.1 Initialisierung der SCORM-Verbindung

Als erstes wird mittels Aufruf der Methode `STDT.Lernbar.SCORM.initializeConnection()` die Verbindung zu SCORM aufgebaut (sofern eine SCORM-API gefunden werden kann). Wie bereits beschrieben, wurde dafür die Bibliothek *SCORM_API_Wrapper.js* verwendet.

6.2.3.2 Initialisierung der TinCanJS

Neben dem Laden der TinCan.js-Datei in der mobile-dev.html muss, um die von TinCanJS bereitgestellten Methoden nutzen zu können, TinCanJS erst einmal im bestehenden Programm instanziiert sowie mit einigen Basis-Informationen versorgt werden. So benötigt TinCanJS zum Senden von Statements eine Angabe des Actors sowie des Endpunktes der Kommunikation (der URL des LRS). Diese Schritte sollen zuerst durchgeführt werden, damit TinCanJS ordnungsgemäß funktioniert und alle Statements

gemäß der Spezifikation der Experience API (siehe 5.3 - Statements) gesendet werden können.

a) TinCanJS instanziieren

Durch den Aufruf, der Funktion `initialize_xAPI()` im Modul `STDT.Lernbar.client` werden alle Informationen zur Nutzung von TinCanJS gesammelt. Dazu gehört die Festlegung der URL des LRS (Endpoint) inklusive einer testweise implementierten einfachen http-Basic-Authentication zum Zugriff auf das LRS sowie der Bildung des Namens der Hauptaktivität, in dessen Kontext alle Statements gesendet werden sollen.

Anschließend wird mit diesen Eigenschaften ein neues Modul `STDT.Lernbar.xAPI` auf Grundlage des TinCanJS-Konstruktors instanziiert.

```
exports.Lernbar.xAPI.init = function (data, callback) {

   exports.Lernbar.xAPI = new _TinCan({
      activity:     { id: data.ROOT_ACTIVITY_ID },
      recordStores: [ data.recordstore ]
   });
   exports.Lernbar.xAPI.TCActive = true;
   [ . . . ]
```

Code 11 - Instanziierung der TinCanJS

Ab dieser globalen Instanziierung der TinCanJS mit den übergebenen Daten in Modul `STDT.Lernbar.xAPI` kann von überall aus auf die von TinCanJS bereitgestellten Methoden und Eigenschaften zugegriffen werden. Durch die Übergabe einer Activity und der RecordStores-Eigenschaft hat die TinCanJS auch schon Initialwerte bekommen, mit denen Sie jedes Statements, das ab jetzt gesendet wird, versehen kann. So ist der Endpoint der Kommunikation (das LRS) schon gesetzt und alle Statements können im Kontext der Hauptaktivität (des einzelnen Kurses) gebildet werden.

Des Weiteren wurde die Instanziierung des TinCanJS-Objektes mit einigen Methoden erweitert, die das Arbeiten mit den Daten des WBTs erleichtern sollen (`setAgent()`, `setAuthority()`, `getContext()`, `saveStatementWithContext()`).

```
exports.Lernbar.xAPI.setAgent = function (agentObj) {
    // analysiert die Daten des Übergebenen agentObj
    // und erstellt daraus einen neuen Actor als Account
    actor.account = {
        homePage: agentObj.homePage,
        name: agentObj.name
    }
    //Neuen Actor aufgrund der Daten erstellen
    STDT.Lernbar.xAPI.actor = new _TinCan.Agent(actor);
}
```

Code 12 – Erweiterung der Instanziierung des Moduls STDT.Lernbar.xAPI mit Methode setAgent()

b) Actor definieren

Weiterhin benötigt das TinCanJS-Objekt Informationen über den Actor, der das WBT ausführt. Der Actor soll dabei, wenn möglich, als Account-Objekt angegeben werden, das ihn eindeutig als Nutzer eines Web-Portals beschreibt (siehe 5.3.1.1 - Actor).

Die Suche nach Informationen über den aktuellen Nutzer des WBTs wird über den Aufruf der Methode `client.initializeUser()` durchgeführt, wobei verschiedene Quellen nach User-Informationen abgesucht werden und, sollten welche gefunden werden, ein Actor-Objekt in `STDT.Lernbar.xAPI.actor` erstellt wird.

1. Sollte eine SCORM-Verbindung vorhanden sein, wird vom LMS der Nutzername des Users abgefragt und in Verbindung mit der URL des LMS als Actor-Account erstellt.

```
function initializeUser(callback) {
   [. . .]
   // SCORM - Userdaten suchen:
   if (STDT.Lernbar.SCORM.API.isFound) {
      var scormUser = STDT.Lernbar.SCORM.getLearner();

      // Actor auf Basis der Daten erstellen:
      STDT.Lernbar.xAPI.setAgent({
         name: scormUser.id,
         homePage: courseHome
      }
   );
   [. . .]
}
```

Code 13 – initializeUser() Beispiel der Suche eines aktiven LMS-/SCORM-Nutzers

2. Besteht keine SCORM-Verbindung, wird nach einer Personalisierung des WBTs in der index.html gesucht *(LernBar-WBTs bieten diese Art der Personalisierung)*.
3. Wird auch dort keine User-Information gefunden, wird unter Verwendung von jQuery.ajax() eine Anfrage an das LernBar Portal gesendet, ob eine aktive Session vorliegt, und ob diese Userinformationen enthält.
4. Existiert auch keine LernBar Portal-Session, so kann keine User-Information gefunden werden. Es wird eine anonyme ID erstellt, aus der der Actor für die Statements gebildet wird. Zusätzlich wird diese ID im *LocalStorage* des Browsers[13] gespeichert, um den User nach einer Unterbrechung des WBTs identifizieren zu können.

c) Authorization definieren

Zur Festlegung der Authority-Eigenschaften der Anfragen wird beispielhaft ein Agent mit Namen „LBC“ und der Homepage http://lernbar.uni-frankfurt.de erstellt. Dies dient allerdings nur zu Testzwecken, da die prototypische Implementierung auf strenge Einhaltung der Autorisierungs-Regeln verzichten wird.

6.2.3.3 Statements mit der Experience API senden

Nach den Schritten, die zur Initialisierung der TinCanJS durchgeführt wurden, kann nun von allen Modulen des WBTs aus auf das Modul `STDT.Lernbar.xAPI` zugegriffen werden, das eine, um einige Methoden erweiterte, Instanz des TinCanJS-Objektes beinhaltet.

[13] LocalStorage ist ein von HTML5 eingeführter Speicherplatz im Browser, in dem Webseiten Daten Session übergreifend zwischenspeichern können. Weitere Informationen siehe [Pi10]

Die wichtigsten Methoden, die das TinCanJS-Objekt bereitstellt, sind dabei `sendStatement()`, `getStatement()`, `sendState()` und `getState()`. Als nächster Schritt sollen mit der sendStatement-Methode Statements über getätigte Aktionen im WBT gesendet werden.

Als Aktivitäten bei denen Statements gesendet werden, wurden dabei die Events „Kursstart“, „Seitenwechsel“ sowie „Frage beantwortet“ gewählt.

Das LernBar WBT verwendet ein sogenanntes PubSub-Modell[14], das es ermöglicht mehrere Methoden auf einmal, bei Eintritt eines Events, aufzurufen. Dabei schreiben sich die Methoden in einzelne Events ein (*subscribe*) und werden, sobald das Event auftritt, von der PubSub-Methode benachrichtigt und ausgeführt (*published*).

Es werden zwei Methoden erstellt, die in das PubSub-Modell eingeschrieben werden.

```
STDT.subscribe({
   message: STDT.Lernbar.message.PAGE_CHANGED,
   callback: pageChanged
});
STDT.subscribe({
   message: STDT.Lernbar.message.QUESTION_ANSWERED,
   callback: questionAnswered
});
```

Code 14 - Verwendung des PubSub-Modells zum Ausführen von Methoden zu bestimmten Events

Die Methoden `pageChanged(message, pageData)` sowie `questionAnswered(message, questionData)` erhalten von der PubSub-Methode alle Daten, die für die Ausführung der Methode benötigt werden.

Aus diesen empfangenen Daten werden von den Methoden Statements generiert, die die Aktion des Lernenden im WBT beschreiben sollen. Eine weitere Methode, die den Kursstart als Statement beschreibt, wird nach Abschluss der Initialisierung aufgerufen.

Damit reagiert der Kurs auf drei verschiedene Aktionen des Nutzers und sendet entsprechende Statements.

Aktion	Statement in der Form
Kursstart	Actor started CourseID
Seitenwechsel	Actor experienced LastPageID in Context of CourseID
Frage beantwortet	Actor answered QuestionID with Result [Score] in Context of CourseID

Weitere Statements zur Beschreibung anderer Aktionen des Nutzers können mit Hilfe der TinCanJS-Bibliothek leicht hinzugefügt werden.

[14] http://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern – Publish/Subscribe-Modell

```
//Wird beim Seitenwechsel im WBT aufgerufen
function pageChanged(message, pageData) {

   //Aufruf der Funktion zum Senden des Statements inklusive Kontext
   //Dieser wird von der Methode automatisch hinzugefügt.
   STDT.Lernbar.xAPI.saveStatementWithContext(
      {
      verb: {
            id: "http://adlnet.gov/expapi/verbs/experienced",
         display: {
               "en-EN": "experienced"
            }
      },
      object: {
         id: ROOT_ACTIVITY_ID + "/" + pageData.id,
         definition: {
            type: "http://adlnet.gov/expapi/activities/page",
            name: {
               "de-DE": pageData.title
            },
         }
      }
      //Die Eigenschaften Actor und Authority werden automatisch
      //von TinCanJS hinzugefügt da diese die Eigenschaften bereit
      //zugeteilt bekommen hat.
   });
}
```

Code 16 – PageChanged-Methode, die ein „Actor experienced Page“ Statement versendet

```
{
   id:  1278181c-79ea-4d0e-b746-8123716257dd,
   timestamp: 2013-09-01T18:15:30.678Z,
   actor:{
      objectType:Agent,
      account:{
         name:       [LBID]123456,
         homePage:   http://lernbar.uni-frankfurt.de/
      }
   },
   verb:{
      id: http://adlnet.gov/expapi/verbs/answered,
      display:{
         de-DE: beantwortet
      }
   },
   object:{
      id:  http://lernbar.uni-frankfurt.de/.de/course.48/question.54,
      objectType: Activity,
      definition:{
         type: http://adlnet.gov/expapi/activities/question,
         name:{
            de-DE: Multiple Choice
         },
      description:{
         de-DE: Welche Komponenten gehören zur LernBar?
      }
   }
   result:{
      score:{
         raw: 2,
         min: 0,
         max: 3
      }
   },
   context:{
      contextActivities:{
         parent:[{
            id:           http://lernbar.uni-frankfurt.de/course.48,
            objectType: Activity
         }]
   }},
}
```

Code 15 - Beispiel eines gesendeten Statements für eine "Frage beantwortet" Aktivität

WBT offline

Sollte das Senden des Statements fehlschlagen, entweder weil das WBT keine Verbindung zum Internet hat, oder das LRS gerade gewartet wird, wird jedes nicht gesendete Statement in einer Warteschlange im *LocalStorage* des Browsers gespeichert (wie in 5.4 - Offline-Phasen des Activity Providers vorgeschlagen). Sobald das erste Statement wieder gesendet werden kann, werden die Statements in der Warteschlange unter Verwendung der `xAPI.saveStatements()` Methode an das LRS übertragen. Somit können kurze Offline-Phasen des WBTs problemlos abgefangen werden ohne Statements zu verlieren.

6.2.3.4 States mit der Experience API senden und empfangen

Zusätzlich zum Senden der Statements wurde die Verwendung der State API zum Speichern des aktuellen Kursstatus im LRS in das WBT integriert.

Dafür wurde zuerst ein neues Objekt `courseState` angelegt, das alle Informationen über einen laufenden Kurs in sich hält. Dieses Objekt wird in dem Modul `STDT.Lernbar.courseStatus` instanziiert.

Beim ersten Kursstart eines User wird ein neues Status-Objekt erzeugt, das Informationen über alle verfügbaren Lektionen, Seiten und Fragen des Kurses enthält.

Durch das schon erwähnte PubSub-Modell wird bei jedem Seitenwechsel das Status-Objekt mit den zuletzt getätigten Aktionen des Nutzers aktualisiert (Timer werden runtergezählt, Seite wird als „LastVisited“ gesetzt, Antworten auf Fragen werden gespeichert, usw…). Nach der Aktualisierung wird das Status-Objekt im JSON-Format und unter einem eindeutigen Namen im LocalStorage sowie im LRS gespeichert.

```
exports.Lernbar.xAPI.setState(
   STATUS_NAME,
   courseStatusObject
   { contentType: "application/json" }
);
```

Code 17 - Beispiel - setState() – Senden eines Kurs-Status im JSON Format an das LRS

Sollte der Nutzer die Bearbeitung des WBTs unterbrechen, so verbleibt das letzte Status-Objekt im LocalStorage sowie im LRS.

Bei einem erneuten Kursaufruf wird innerhalb der Methode `client.initializeStatus()`, die auch von `client.initialize()` aufgerufen wird, geprüft, ob ein Status-Objekt im LRS oder im LocalStorage vorhanden ist.

```
//State aus dem LRS laden
serverStatus = STDT.Lernbar.xAPI.getState(STATUS_NAME);
//State aus dem LocalStorage laden
localStatus = localStorage.getItem(STATUS_NAME);
```

Code 18 - Laden des State vom LRS und dem LocalStorae des Browsers

Sollten ein Status-Objekt aus der State API und ein Status-Objekt aus dem LocalStorage vorliegen, so wird der TimeStamp der beiden verglichen und das jeweils aktuellere zur

Initialisierung des restlichen Kurses verwendet. Somit wird durch die zusätzliche Speicherung im LocalStorage Offline-Phasen des Kurses entgegengewirkt und theoretisch ist das WBT damit sogar Offline nutzbar.

6.2.4 Mögliche Erweiterungen des xAPI-WBT-Prototyps

Der erstellte Prototyp ist in der Lage, Statements zu drei verschiedenen Aktionen des Nutzers zu generieren und an das LRS zu senden sowie den Status des Kurses als State an das LRS zu senden und abzufragen. Diese Funktionen stellen die grundlegenden Funktionen dar, die durch die Implementierung der Experience API in ein WBT umgesetzt werden können. Weitere Funktionen, die über die Anforderungen einer prototypischen Implementierung hinausgehen, sind jedoch denkbar und sollen nun kurz vorgestellt werden, um die Möglichkeiten, die ein xAPI-WBT bieten kann, darzustellen.

Verwendung verschiedener LRS

Im erstellten xAPI-WBT-Prototyp ist die URL des verwendeten LRS fest vorgegeben. Durch eine Erweiterung der Implementierung wäre es allerdings möglich, das zu verwendende LRS in einer der Konfigurations-Dateien des WBTs abzulegen und dort auszulesen. Die TinCanJS-Bibliothek bietet dabei sogar die Möglichkeit, die Statements an mehrere LRS gleichzeitig zu senden. So könnte Autoren durch eine Anpassung im Autorenprogramm (zum Beispiel dem LernBar Studio) die Möglichkeit gegeben werden, alternative Learning Record Stores anzugeben, sodass die Statements im LRS ihrer Wahl, oder sogar in mehreren LRS gleichzeitig, gespeichert werden können.

Verwendung der Activity Profile API

Würde jede Frage, die der Nutzer in einem WBT bearbeitet, zusätzlich Daten in das entsprechende Activity Profile übertragen, könnte man dem Nutzer des WBTs nach seiner Antwort einen Durchschnittswert der erreichten Punkte oder die durchschnittliche Bearbeitungszeit aller anderen Nutzer anzeigen. Damit wäre es möglich, dass sich der Nutzer mit anderen Teilnehmern vergleichen kann.

Zusätzlich könnte ein Activity Profile es möglich machen, dass sich Teilnehmer innerhalb eines WBTs vernetzen können. So könnte jede Activity (gesamter Kurs oder einzelne Seite/Frage) durch eine Kommentarfunktion oder einen Chat ausgestattet werden, die es den Teilnehmern erlaubt, miteinander über die Aktivität zu diskutieren oder Anmerkungen zu veröffentlichen.

Verwendung der Agent Profile API

Unter Voraussetzung, dass das Sicherheitsmodell der Experience API im LRS umgesetzt wurde[15], könnte man am Ende eines jeden LernBar-WBTs eine spezielle

[15] Siehe 5.3.1.10 - Authority

Auswertungsseite entwickeln, die dem Nutzer seine, in seinem Agent Profile gespeicherten Aktivitäten, anzeigt.

Dies könnte dazu dienen, dem Nutzer einen Überblick über seinen Fortschritt in voneinander abhängigen WBTs zu visualisieren oder ihn auf weitere, noch nicht bearbeitete WBTs, hinzuweisen.

Auch eine kursübergreifende Gesamtauswertung wäre realisierbar.

Erfassung von Aktivitäten außerhalb des WBTs, aber in dessen Kontext

Auf http://tincanapi.com/bookmarklet/ hat Rustici Software ein *Bookmarklet*[16] bereitgestellt, das es möglich macht, jede besuchte Internetseite in einem LRS zu protokollieren. Auf diesem Konzept aufbauend wäre es denkbar, dass das WBT eine Funktion bereitstellt, die dem Nutzer ein dynamisch generiertes *Bookmarklet* bereitstellt.

Bei Ausführung des Bookmarklets könnte ein Statement über die aktuell besuchte Internetseite des Nutzers an das LRS übertragen werden. Dazu müssten im dynamisch generierten Bookmarklet die Agent-Daten des Nutzers, das Verb (*experienced* zum Beispiel) sowie das WBT, in dem das Bookmarklet generiert wurde, als Kontext des Statements fest implementiert sein. Das Object des Statements wäre dann die aktuell aufgerufene Internetseite des Nutzers. Somit könnten Autoren den WBT-Teilnehmern zum Beispiel die Aufgabe stellen, Informationen über ein bestimmtes Thema einzuholen und die Teilnehmer dazu auffordern, diese Suche durch Protokollierung der besuchten Seiten festzuhalten.

6.2.5 Fazit der Implementierung der Experience API in ein WBT

Der erstellte Prototyp ist in der Lage, Statements sowie den Status des Kurses als State an das LRS zu senden und abzufragen. Somit kann die Implementierung der Experience API in ein WBT als gelungen bewertet werden. Maßgeblich dafür verantwortlich ist die Verwendung der TinCanJS-Bibliothek, die Entwickler dabei unterstützt, die Kommunikation mit dem LRS durchzuführen, sowie die Statements nach Experience API Spezifikation zu generieren.

Zusammengefasst kann man die nötigen Arbeitsschritte zur Implementierung der Experience API in ein bestehendes WBT aufzählen als:

1. TinCanJS instanziieren & initialisieren.
2. Daten des Kurses sammeln, sofern noch nicht vorhanden.
3. Mit Unterstützung von TinCanJS Statements generieren.
4. Statements mit TinCanJS senden und empfangen.

[16] Als Bookmark gespeicherte JavaScript-Funktion, das auf jeder Webseite aufgerufen werden kann.

Handelt es sich dabei um ein weniger komplexes WBT, mit weniger Funktionen als das verwendete, würde sich die Implementierung sicher noch einfacher gestalten. Beim vorliegenden LernBar-WBT wurde dabei viel Arbeit darauf verwendet, die modularisierte Struktur des JavaScript-Codes zu verstehen sowie die benötigten Daten (Status-Objekt) zu generieren.

Eine Implementierung der Experience API in ein WBT muss nicht zwangsläufig mit Hilfe der TinCanJS geschehen. So können die Statements auch „frei" als JSON-Objekte generiert und per *XMLHttpRequest* oder per *jquery.ajax()* an das LRS gesendet werden.[17] Jedoch ist die Vorab-Validierung der einzelnen Eigenschaften der Statements durch TinCanJS eine enorme Hilfe für den Entwickler, um die von der Experience API aufgestellten Regeln über die genaue Form der Daten einzuhalten. Jeder Fehler in der Syntax der Statements wird sofort bemerkt und, sollte er nicht automatisch korrigiert werden, bemängelt. Die bereitgestellten get- und set-Statement- sowie State-Methoden ermöglichen eine zuverlässige Kommunikation mit dem LRS, ohne sich vorher mit den Eigenschaften von http-Request-Headern oder andern technischen Eigenschaften von http-Requests befassen zu müssen.

Tests auf verschiedenen Geräten haben gezeigt, dass die TinCanJS-Bibliothek auch auf mobilen Geräten (iPhone in verschiedenen Versionen und verschiedene Android Geräte) sowie in den verschiedenen Betriebssystem<->Browser Variationen (Windows XP, 7, 8 sowie Internet Explorer, Google Chrome, FireFox) keine Probleme bereitet und fast immer Statements und States senden sowie empfangen konnte.

Die in Kapitel 5.2.1.5 beschriebenen Einschränkungen zur Nutzung von *CORS* stellen somit auch die Grenzen der Browser-Kompatibilität der TinCanJS dar (siehe Abbildung 21).

Die TinCanJS-Bibliothek ist noch nicht final, so bietet sie zum Beispiel noch keine Unterstützung für den oAuth-Authentifizierungs-Prozess, wodurch die Sicherheitsfeatures der Experience API[18] nur in Ansätzen umgesetzt werden konnten. Eine einfache http-Basic-Authentication mit Benutzername und Passwort wurde testweise implementiert und wird zum Zugriff auf das LRS durch das WBT durchgeführt. Dies sichert gegen einen unbefugten Zugriff auf das LRS in Grundzügen ab. Es ist allerdings höchst wahrscheinlich, dass TinCanJS in zukünftigen Versionen auch das Sicherheitsmodell der Experience API besser umsetzen wird und es ist davon auszugehen, dass dieses dann genauso komfortabel zu verwenden sein wird, wie die bereits implementierten Funktionalitäten. Ebenso verhält es sich mit der Activity Profile API und der Agent Profile API. An diesen wird zum Zeitpunkt der Implementierung noch gearbeitet.

[17] Siehe Kapitel 6.2.2 - Verwendete Techniken zur Implementierung der Experience API
[18] Siehe Kapitel 5.3.1.10 - Authority

Einziger Nachteil der TinCanJS-Bibliothek ist ihre beachtliche Größe und Komplexität. Dabei handelt es sich sogar bei der komprimierten Version der TinCanJS (ohne Kommentare und nicht programmrelevanten Inhalte) um eine 66 Kilobyte große Datei. Stellt man dagegen die Größe eines einfachen WBTs mit einfacher Navigation durch einzelne Seiten, so könnte die TinCanJS für kleinere Projekte überdimensioniert sein.

➔ Die prototypische Implementierung der Experience API in ein bestehendes WBT konnte ohne größere Probleme durchgeführt werden. Die Spezifikation der Experience API kann somit aus Sicht eines WBT-Entwicklers als umsetzbar beschrieben werden. Alle von der Experience API Spezifikation geforderten Regeln konnten eingehalten werden und sind mit HTML5, JavaScript und der TinCanJS Bibliothek umsetzbar. Der Entwicklungsprozess wurde dabei durch die Verwendung der TinCanJS-Bibliothek vereinfacht, da diese alle Hauptfunktionen, die zur Umsetzung benötigt werden, bereitstellt. Somit kann man die Verwendung der TinCanJS für alle Implementierungen, die eine JavaScript-Bibliothek verwenden können (WebApps, Webseiten, hybride Apps), empfehlen.

6.3 Entwicklung eines Learning Record Stores

Nachdem die Experience API auf Client-Seite in einen Activity Provider (WBT) integriert wurde, soll nun auch auf Server-Seite eine prototypische Implementierung eines einfachen Learning Record Store (LRS) umgesetzt werden.

Dafür werden erst die zur Umsetzung verwendeten Techniken beschrieben und anschließend die prototypische Umsetzung vorgestellt und besprochen.

6.3.1 Verwendete Techniken zur Implementierung des LRS

Zuallererst soll untersucht werden, ob schon bestehende Implementierungen oder Frameworks für die Entwicklung von LRS existieren, und ob diese als Grundlage der Implementierung verwendet werden können. Anschließend werden die zur Umsetzung gewählten Techniken beschrieben.

6.3.1.1 Existierende Implementierungen von LRS

SCORM Cloud

Rustici Software, welche das Project TinCan durchgeführt hat, bietet unter dem Namen *SCORM Cloud*[19] einen Service an, der es erlaubt, SCORM- und Experience API–Inhalte im Internet zu speichern und die daraus empfangen Daten zu visualisieren und weiterzuverarbeiten. Dabei ist in der SCORM Cloud seit Kurzem auch ein LRS integriert.

[19] http://scorm.com/scorm-solved/scorm-cloud-features/

Der Service ist kommerziell, allerdings existiert die Möglichkeit einen Testzugang zu erstellen, um die bereitgestellten Funktionalitäten zu testen.

Das SCORM Cloud-LRS wurde mit Hilfe eines solchen Testzugangs dazu verwendet, um während der Entwicklung des WBT-Prototyps die Korrektheit der Statements zu überprüfen. So ist durch die Beteiligung von Rustici Software am Project TinCan davon auszugehen, dass das in der SCORM Cloud integrierte LRS alle Experience API Spezifikationen korrekt umsetzt. Dies hat sich während der Verwendung des LRS auch bestätigt.

Weiterhin bietet Rustici-Software auf seiner Homepage kurze Anleitungen und Hilfestellungen für die Entwicklung eines eigenen LRS an. [20]

Ebenso existieren weitere LRS-Implementierungen, die zum Beispiel auf http://tincanapi.com/adopters/ aufgelistet werden. Bei den meisten davon handelt es sich jedoch um kommerzielle Produkte, sodass keines dieser LRS bei der Entwicklung des prototypischen LRS als Vorlage dienen kann.

LRS-Prototyp der ADL (Python und MySQL)

Die einzige, frei verfügbare Implementierung eines LRS stammt direkt von der ADL und kann unter https://github.com/adlnet/ADL_LRS heruntergeladen werden. Dabei wird als Programmiersprache Phyton verwendet und als Datenbank, zur Speicherung der Statements, eine MySQL-Datenbank.

Der Ersteindruck des LRS ist allerdings nicht sehr gut. So befand sich der Prototyp der ADL bei Beginn dieser Arbeit noch in einer frühen Entwicklungsphase. Außerdem sollte der in dieser Arbeit entwickelte LRS-Prototyp eventuell als Grundlage eines LRS für das LernBar Portal, das mit PHP umgesetzt wurde, dienen.

Aus diesen und aus den im Folgenden beschriebenen Gründen der Speicherung der Daten, wurde von einer Verwendung des, von der ADL bereitgestellten Prototyps, abgesehen und somit eine komplette Eigenentwicklung des LRS-Prototyps durchgeführt.

6.3.1.2 Gewählte Techniken

Wie in der Spezifikation der Experience API beschrieben, handelt es sich bei dem LRS um einen RESTful WebService, der Daten in Form von Ressourcen im Internet beschreibbar und abrufbar machen soll.[21]

Somit benötigt das LRS eine Technik, um als RESTful WebService implementiert zu werden und zusätzlich ein Datenbank-Modell, dass das Speichern und Abrufen der übertragenen Daten möglichst unkompliziert ermöglicht.

[20] http://tincanapi.com/building-a-learning-record-store/ und https://github.com/RusticiSoftware/launch/blob/master/lms_lrs.md

[21] Siehe 5.2 - Data Transfer APIs

RESTful WebServices mit dem slim-Micro-Framework

Zur Entwicklung von RESTFul WebServices auf Basis von PHP existiert eine Vielzahl von Frameworks, die dem Entwickler verschiedene Methoden zur Vereinfachung der Implementierung bereitstellen.[22]

Auch für die Entwicklung des LRS-Prototyps soll ein solches Framework verwendet werden, um die Implementierung zu erleichtern. Dabei fiel die Wahl des verwendeten Frameworks auf das slim-Framework.[23]

Das slim-Framework ist ein sogenanntes *Micro-Framework,* das unter der MIT Lizenz vertrieben wird und verschiedene Methoden bereitstellt, um die Umsetzung eines RESTFul WebService zu ermöglichen. Dabei stellt das slim-Framework, im Gegensatz zu „vollen" Frameworks wie *Recess* oder dem *Zend Framework*[24]*,* nur Methoden für die Grundfunktionalitäten eines RESTful WebServices bereit.

Das slim-Framework bietet dabei hauptsächlich die Möglichkeit, sogenanntes *Routing* umzusetzen. Darunter versteht man eine Art Umleitung, die durchgeführt wird, sobald ein Client einen http-Request an den Server stellt. Der vom Client gestellte http-Request an eine bestimmte Ressource wird vom slim-Framework intern auf eine bereitstehende php-Methode umgeleitet und alle übertragenen Daten des http-Requests der Methode zur Verfügung gestellt. Die mit der Ressource verknüpfte php-Methode kann dann die Antwort auf den http-Request des Clients generieren und mittels des slim-Frameworks an den Client ausliefern.

Zusätzlich besteht die Möglichkeit *Middleware* einzubinden. Middleware stellt dabei php-Methoden dar, die bei jedem eingehenden http-Request vor der eigentlichen Verarbeitung des Requests vorgeschaltet werden. So könnte zum Beispiel durch Verwendung der geeigneten Middleware eine Authentifizierungs-Kontrolle vor jeder Verarbeitung eines Requests durchgeführt werden.

Da diese bereitgestellten Funktionen für die Entwicklung eines schlanken, einfachen LRS ausreichen, wird die prototypische Entwicklung des LRS mit dem slim-Framework umgesetzt.

Speicherung der Daten in einem nicht-relationalen Datenbank-Modell - MongoDB

Wie eingangs beschrieben, verwendet der von der ADL bereitgestellte LRS-Prototyp eine MySQL-Datenbank, um die übermittelten Statements und Daten zu speichern und zu verwalten.

[22] http://blog.programmableweb.com/2011/09/23/short-list-of-restful-api-frameworks-for-php/ - Eine kurze Auflistung verschiedener Frameworks zur Umsetzung von RESTful WebServices

[23] http://www.slimframework.com/

[24] http://www.recessframework.org/ und http://framework.zend.com/

Betrachtet man jedoch Protokolle aus den Entwickler-Meetings der Experience API (damals noch Project TinCan), stößt man auf folgenden interessanten Beitrag von einem ADL-Mitarbeiter zur Umsetzung des LRS-Prototyps:

> *„[…] we are using MySQL and Django/Python – relational DB instead on MongoDB to verify a reference LRS implementation“*[25]

Geht man noch ein bisschen weiter zurück, so findet man im GitHub von Rustici Software einen weiteren LRS-Prototyp aus den Anfangszeiten des Project TinCan. Dieser verwendet eine MongoDB-basierte Datenbank zur Speicherung der Daten.[26] Der Prototyp wurde allerdings seit 2011 nicht mehr aktualisiert und kann somit ohne große Änderungen nicht verwendet werden. Jedoch kann man aus der Existenz eines solchen Prototyps ableiten, dass die Form der Statements unter der Annahme der Verwendung eines dokumentbasierten nicht-relationalen Datenbank-Modells wie MongoDB entwickelt wurde.

Eine MongoDB-Datenbank (oder nicht-relationale Datenbanken im Allgemeinen) verwendet im Gegensatz zu einer MySQL-Datenbank keine Tabellen und Spalten, um Daten zu verwalten, sondern *Collections*, in denen die kompletten Dokumente, vorzugsweise im JSON-Format, abgespeichert und abgerufen werden können. Somit stellt MongoDB eine dokumentbasierte nicht-relationale Datenbank dar.

Suchanfragen in der Datenbank werden durch Kriterien durchgeführt, nach denen die vorhandenen Dokumente gefiltert werden. Dabei handelt es sich bei den Suchkriterien wiederum um JSON-Objekte, die Eigenschaften beinhalten, nach denen alle vorhandenen Dokumente gefiltert werden sollen.[27]

Beispiel: Angenommen, es wird eine Collection „Statements“ angelegt, in die die übermittelten Statements, ohne Veränderung, als komplette Dokumente abgelegt werden. Wird eine Suchanfrage mit dem Kriterium `{ actor.name: „Klaus“ }` ausgeführt, werden von MongoDB alle Statements zurückgegeben, die genau diese Eigenschaft mit dem angegebenen Wert beinhalten.

Die Form der Daten, die von einer MongoDB zur Datenspeicherung sowie Suche verwendet wird, stimmt somit mit der Form der Daten die, von einem Activity Provider an eine der vier APIs des LRS[28] gesendet werden, überein.

Die Form der Statements als vollständige, zustandslose JSON-Objekte, die jederzeit alle Eigenschaften beinhalten, die das Statement beschreiben, macht jedes Statement zu einem vollständigen Dokument, das in einer MongoDB abgelegt werden kann. Die Definition des http-GET-Statement-Requests[29], der alle Statements abfragt, die mit dem

[25] http://tincanapi.wikispaces.com/Minutes+-+April+19,+2012
[26] https://github.com/RusticiSoftware/TinCan_Prototypes/tree/baa
[27] Mehr Informationen zum Umgang mit MongoDB in [Ch13] und [Ba12]
[28] Siehe 5.2.2 - Die vier APIs der Experience API
[29] Siehe 5.2.2.1 - Statement API

übertragenen JSON-Objekt übereinstimmen, kann auch ohne Zwischenschritte direkt in einer MongoDB ausgeführt werden.

Diese Übereinstimmungen sind keineswegs zufällig. So wurde die dokumentbasierte-NoSQL-Datenbank MongoDB unter anderem explizit für den Einsatz mit RESTful WebServices und JSON-Objekten konzipiert. Da es sich bei dem LRS um einen RESTful WebService handelt, ist die Verwendung von MongoDB also durchaus möglich und erscheint sinnvoll.

Ein Nachteil von MongoDB besteht in einem höheren Speicherbedarf der Datenbank, mit der der komfortable Umgang mit den Daten erkauft wird. Im Vergleich zu MySQL wird, je nach Varianz der Eigenschaften der Daten, die ungefähr eineinhalb bis dreifache Menge an Speicher benötigt, um die gleiche Performance wie bei einer MySQL Datenbank zu erreichen. Somit könnte für größere LRS, die mit einem sehr hohen Datenaufkommen rechnen müssen, die Verwendung einer relationalen Datenbank durchaus die bessere Alternative darstellen. Ein weiterer Nachteil besteht darin, dass eine MongoDB keine JOINs unterstützt. Dies führt dazu, dass komplexe Abfragen nur schwer auszuführen sind, oder dass zuerst alle beteiligten Datensätze ausgelesen werden müssen, um danach im Client zusammengesetzt werden zu können.

Für die prototypische Umsetzung allerdings wird in Anbetracht der geringen Datenmengen der Statements eine MongoDB als Datenbank verwendet.

6.3.2 Implementierung des LRS-Prototyps

Nachdem die für die Umsetzung des LRS-Prototyps verwendeten Techniken gewählt und beschrieben wurden, soll nun die Implementierung des LRS vorgestellt werden.

Die Implementierung des LRS-Prototyps fand auf einem Apache-WebServer mit PHP 5.3 statt. So soll zuerst die Einrichtung der benötigten Server-Erweiterungen besprochen werden und anschließend die Implementierung kurz vorgestellt werden.

6.3.2.1 .htaccess

Ein neues Verzeichnis `/lrs/`, das den Endpoint für das LRS darstellen soll, wurde auf dem Server angelegt. Um alle Anfragen an Ressourcen des LRS an eine einzige php-Datei umzuleiten, wird das Verzeichnis mit einer `.htaccess`-Datei ausgestattet, die diese Umleitung realisiert.

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ lrs.php [QSA,L]
```

Code 19 - .htaccess Datei zur Umleitung aller Serveranfragen an die Datei lrs.php

6.3.2.2 lrs.php anlegen

Eine Datei `lrs.php` wurde erzeugt und im Verzeichnis `/lrs/`, in dem sich auch schon die .htaccess-Datei befindet, abgelegt. Um die http-Requests, die von den Activity Providern gesendet werden, zu akzeptieren, müssen einige Access-Control-Header gesetzt werden, sodass sie den Anfrage-Headern entsprechen. Die wichtigste Header-Variable ist dabei `Access-Control-Allow-Origin`. Diese muss auf `*` gesetzt werden, sofern man alle Anfragen, egal von welcher Domain sie stammen, akzeptieren will. Dies stellt die serverseitige Umsetzung von CORS[30] dar.

```
//CORS aktivieren, Anfragen werden von überall akzeptiert
header('Access-Control-Allow-Origin: *');
```

Abbildung 25 - CORS auf dem Server aktiveren

6.3.2.3 MongoDB einrichten

Zur Nutzung der MongoDB als Datenbank für das LRS muss der MongoDB-Server zuerst auf dem WebServer installiert werden. Anleitungen für die Installation auf verschiedenen Server-Betriebssystemen findet man unter http://docs.mongodb.org/manual/installation/.

PHP bietet in Version 5.3 keine native Unterstützung für die Nutzung einer MongoDB Datenbank. Somit muss zusätzlich der *MongoDB PHP Driver* von http://docs.mongodb.org/ecosystem/drivers/php/ installiert werden, um von PHP aus auf eine MongoDB-Datenbank zugreifen zu können. Die API des MongoDB PHP Drivers ist unter http://php.net/manual/en/book.mongo.php verfügbar.

Sind MongoDB-Server sowie Driver installiert, ist die Installation abgeschlossen und es kann über PHP-Aufrufe oder über graphische Administrations-Tools wie RockMongo[31] auf Datenbanken und Collections zugegriffen werden.

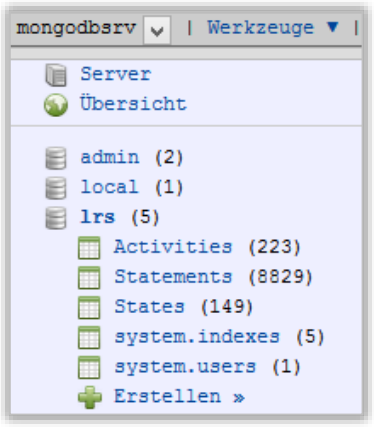


Abbildung 26 - Erstellte MongoDB Datenbank

Mit Hilfe von RockMongo wurde eine Datenbank „lrs“ erstellt mit den Collections „Activities, „Statements“ und „States“, in die die vom Activity Provider gesendeten Daten gespeichert werden sollen.

In der `lrs.php` kann nun durch die Instanziierung `$m = new Mongo($connection_url)` eine Verbindung zur MongoDB hergestellt werden.

[30] Siehe 5.2.1.5 - Cross-Origin Ressource Sharing
[31] http://rockmongo.com/

6.3.2.4 Routings mit dem slim-Framework einrichten

Nachdem die Einrichtung des Servers abgeschlossen ist, die Hauptdatei, die zur Implementierung des LRS verwendet werden soll, angelegt ist und die Verbindung zur MongoDB hergestellt wurde, wird nun das slim-Framework initialisiert.

Dafür muss das slim-Framework von http://www.slimframework.com/ heruntergeladen werden und in einem Unterverzeichnis des LRS entpackt werden (`/lrs/Slim/`). Anschließend kann im php-Skript das Framework eingebunden und instanziiert werden.

```php
//Slim-Framework einbinden.
require 'Slim/Slim.php';
use Slim\Slim;

Slim::registerAutoloader();
$app = new Slim();
```

Abbildung 27 - Laden und Initialisieren des Slim-Frameworks

Nach der Instanziierung in der Variable `$app` können alle Funktionen des slim-Frameworks verwendet werden.

So können nun sehr einfach die Routings erstellt werden, die die Anfragen an die bereitgestellten Ressourcen auf die implementierten Methoden umleiten.

Dies wird nun am Beispiel eines ankommenden http-POST-Requests an die Ressource `lrs/statements/` beschrieben:

```php
//Routing Regel:
//Ein ankommender http-POST-Request an die URL /statements
//wird von slim an die Funktion „post_statement“ weitergeleitet
$app-> post('/statements', 'post_statement');

//Ein Statement in die MongoDB eintragen
function post_statement()
{
    //Zur MongoDB-Datenbank verbinden
    $db = conn_mongo();
    //Hole die empfangenen Daten aus der Post-Data
    $app = Slim::getInstance();
    //Den kompletten PayLoad des Requests holen.
    $statement = $app->request()->getBody();
    //Die JSON-Daten in ein PHP-Array umwandeln
    $statement = json_decode($statement, true);

    //Die Collection Statements in der DB auswählen
    $collection = $db->selectCollection("Statements");
    //Die Daten in die Datenbank eintragen
    $collection->insert($statement);

    //Keine Daten zurückgeben, nur den http-Status 204
    //(wie in xAPI-Specs gefordert)
    $app->response()->status(204);

    // Das Objekt des Statements, wenn es eine Activity ist
    // in einer eigenen Collection zusätzlich speichern
    if (isset($statement["object"])) {
        save_Activity($statement["object"]);
    }
}
```

Code 20 – Verarbeitung eines http-Requests mit slim am Beispiel von POST lrs/statements

Dabei handelt es sich bei der post_statement-Methode um die einfachste der implementierten Methoden. Weitere erstellte Routings für http-Requests und somit erreichbare und durch Methoden implementierte Ressourcen sind:

Ressource	Typ	Aktion
/statements	GET	Sucht Statements in der Datenbank die den Suchkriterien entsprechen, und gibt die Statements zurück
	POST	Speichert ein oder mehrere Statements in der Datenbank
/activities/state	GET	Sucht einen State aus der Datenbank der zu den Suchkriterien passt, und gibt ihn an den Client zurück
	POST	Upsert eines States in der Datenbank
	DELETE	Löscht einen State aus der Datenbank
/getPortalUsername	GET	Prüft die Server-Session nach einem aktiven Login und gibt die Nutzerdaten zurück

Es wurden alle LRS-Ressourcen, die Statements und States erreichbar machen, implementiert und den Experience API Spezifikationen entsprechend umgesetzt. Das LRS verfügt damit über einige der Grundfunktionalitäten, die ein LRS, nach Experience API Spezifikation, bereitstellen muss. Dies soll im Rahmen dieser Arbeit genügen, um eine Aussage über die Implementierbarkeit eines LRS treffen zu können.

6.3.2.5 Sicherheit der Anfragen

Auf eine Anwendung der Authority Regeln, wie sie in der Experience API beschrieben wurden[32], wurde wegen der fehlenden oAuth-Implementierung in der TinCanJS-Bibliothek im Client verzichtet.

Allerdings wurde testweise eine Middleware zur Basic-http-Authentication in das slim-Routing eingefügt, mit der die Identität des Activity Providers über die gesendeten Authorization-Header des http-Requests, geprüft wird. So kann sichergestellt werden, dass nur Activity Provider, die über den Usernamen und das Passwort verfügen, Anfragen senden können.

Da der Authentifizierungs-Hash im Client jedoch durch simples Debuggen des JavaScript-Codes ausgelesen werden kann, trägt diese Lösung zwar nur minimal zur Steigerung der Sicherheit des Servers bei, aber es soll zeigen, dass das Einfügen einer Middleware die geeignete Lösung zur Authentifizierung des Clients und Users darstellen kann.

So könnte, wenn eine User- und Client-Liste eingefügt wird, in einer entsprechenden Middleware vor jedem Routing die Authority-Eigenschaft des übermittelten Statements oder des States überprüft werden und nur bei gültiger Autorisierung ausgeführt werden.

[32] Siehe 5.3.1.10 - Authority

Anstatt einer eigenen Implementierung einer User- und Rechteverwaltung ist auch die Integration des LRS in ein bestehendes Web-Portal, das bereits Funktionen zur Authentifizierung bereitstellt, denkbar.

So wurden im prototypischen WBT schon die Funktionen implementiert, um den User, der das WBT verwendet, durch Auslesen der Userdaten von einer SCORM-API, die ein LMS bereitstellt oder durch die Prüfung nach einer aktiven Session des LernBar-Portals, zu identifizieren.

Am Beispiel des LernBar-Portals könnte man das LRS als eigenen Service in das LernBar Portal integrieren und die schon bestehende Rechteverwaltung verwenden, um die Authentifizierung durchzuführen. Registrierten Autoren und Nutzern könnte im LernBar Portal der Zugang zu Reporting-Tools gegeben werden, mit denen Sie die letzten Aktivitäten in ihren absolvierten oder bereitgestellten Kursen verfolgen könnten. Weiterhin könnte eine Funktion zur Erstellung neuer API-Keys angeboten werden, mit der weitere Applikationen zur Nutzung des LRS registriert werden können. Damit wäre auch die Verwendung von externen Analyse-Tools, die auf die Ressourcen des LRS zugreifen könnten, möglich.

6.3.2.6 Statement Viewer

Zur Überprüfung der Funktion des LRS wurde unter anderem ein von Rustici Software bereitgestellter Prototyp eines Statement Viewers verwendet.[33]

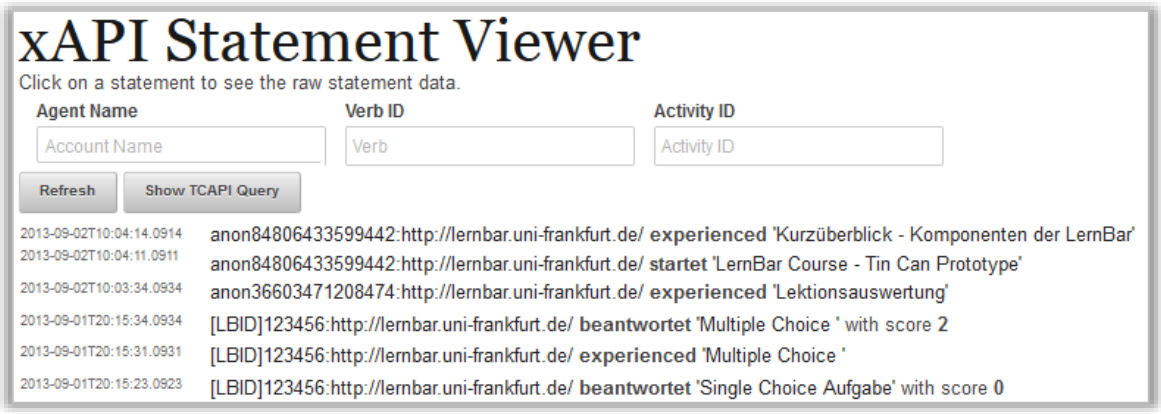


Abbildung 28 - Beispiel: Ausgabe des Statement Viewers

Der Statement Viewer wurde als HTML-Seite umgesetzt und verwendet ebenso wie der erstellte WBT-Prototyp die TinCanJS Bibliothek, um mit dem LRS kommunizieren zu können. Der Statement Viewer wurde leicht angepasst und auf das erstellte LRS konfiguriert. Er kann dafür verwendet werden, die letzten gesendeten Statements vom LRS abzurufen und zu betrachten, welche Aktionen als letztes ausgeführt wurden. Zusätzlich können Filter nach Actor.name, Activity.ID und Verb.ID angegeben werden, die die Suchanfragen an das LRS steuern.

[33] https://github.com/RusticiSoftware/TinCanStatementViewer - Statement Viewer Prototyp

Damit konnte eine korrekte Funktionsweise der getStatement-Methode überprüft werden.

6.3.3 Fazit der prototypischen Implementierung eines LRS

Die prototypische Implementierung des LRS auf Basis von PHP und dem slim-Framework sowie einer MongoDB-basierenden Datenbank zur Speicherung der anfallenden Daten konnte erfolgreich umgesetzt werden.

Dabei setzt die prototypische Implementierung des LRS die State API und die Statement API in ihren Grundfunktionen (Senden und Empfangen von Statements und States) entsprechend der technischen Spezifikation der Experience API um. Dabei konnten keine Probleme bei der Umsetzung der Experience API mit den ausgewählten Techniken festgestellt werden.

Die Verwendung einer MongoDB-basierten Datenbank zur Verwaltung der Daten erwies sich als sehr effizient in Verbindung mit dem, als RESTful WebService, konzipierten LRS. So war der Zugriff auf die Daten, dank der speziell auf die Anforderungen von RESTful WebServices zugeschnittene MongoDB, verhältnismäßig einfach umzusetzen. Ein mühseliges Erstellen von geeigneten Datenbankstrukturen, wie es bei Verwendung von MySQL notwendig gewesen wäre, musste nicht durchgeführt werden. Da auch sonst keinerlei Probleme bei der Implementierung festgestellt wurden, kann die Wahl der Verwendung einer MongoDB als Datenbank für ein LRS empfohlen werden.

Das slim-Framework stellte die benötigten Methoden zur Definition von Routing-Regeln bereit, mit denen die vom LRS bereitzustellenden Ressourcen implementiert werden konnten. Für Entwickler, die bereits Erfahrungen in der Entwicklung von RESTful WebServices haben, sollte eine Implementierung eines LRS größtenteils, mit schon vorhandenem Wissen, umgesetzt werden können.

Tests in Verbindung mit dem angepassten Statement Viewer sowie des WBT-Prototyps haben gezeigt, dass das LRS von überall zu erreichen ist und Statements sowie States zuverlässig in ihm gespeichert sowie abgerufen werden können. Dabei war der Zugriff auf das LRS von mobilen Geräten auf Basis von Android oder iOS sowie von verschiedenen Desktop-Browser problemlos möglich.

➔ Eine Implementierung eines LRS ist mit aktuell verfügbaren Techniken möglich. Auch die Einhaltung aller in der Experience API Spezifikation aufgestellten Regeln ist durch die gewählten Techniken umsetzbar.

6.4 Fazit der prototypischen Umsetzung

Nachdem die prototypischen Implementierungen in das WBT sowie des LRS vorgestellt wurden, sollen die Ergebnisse nun zusammengefasst und unter Betrachtung der in Kapitel 4.3.2 aufgestellten Ziele der Experience API bewertet werden.

Das erstellte, prototypische xAPI-WBT kann Statements über getätigte Aktivitäten des Nutzers an die Statement-API eines LRS senden, sowie seinen aktuellen Kurs-Status in der State-API speichern und bei Kursstart abrufen. Dabei entsprechen alle gesendeten Statements der Experience API Spezifikation.

Das entwickelte prototypische LRS stellt die Grundfunktionalitäten der Statement- sowie State-API bereit. So können Statements und States an das LRS gesendet und von ihm abgerufen werden. Dies wurde mit dem erstellten xAPI-WBT-Prototyp sowie einem einfachen Statement Viewer getestet und alle entwickelten Funktionen des LRS verhalten sich so, wie von der Spezifikation der Experience API gefordert.

➔ Es wurde ein Experience API–kompatibles WBT sowie ein einfaches LRS entwickelt, die Experience API-konforme Daten generieren, versenden sowie empfangen und verarbeiten können. Somit konnte gezeigt werden, dass eine Implementierung der Experience API, mit aktuell verfügbaren Techniken, möglich ist.

Flexibles Daten-Modell ✓

Durch die Verwendung der TinCanJS-Bibliothek im xAPI-WBT-Prototyp werden alle Anforderungen der Experience API Spezifikationen an die Form der Statements und States genau eingehalten. Da schon die Analyse der Spezifikation der Experience API ergeben hat, dass die Form der Statements sehr flexibel zur Darstellung von Aktivitäten aller Art verwendet werden kann, gilt dies auch für den xAPI-WBT-Prototyp. Es wurden hierbei zwar nur einfache Statements generiert und übertragen, die Möglichkeit auch komplexere Aktivitäten zu beschreiben, ist allerdings durch die TinCanJS gegeben.

Im LRS-Prototyp werden durch die Implementierung keine zusätzlichen Beschränkungen in der Form der Daten benötigt. Die verwendete MongoDB-basierte Datenbank akzeptiert alle vom Client gesendeten Daten nach Experience API Spezifikation. Dabei werden die empfangenen Daten fast unverändert in der Datenbank gespeichert und somit gelten auch hier die Ergebnisse der Analyse der Experience API.

➔ Das Experience API Datenmodell kann in den Implementierungen des xAPI-WBTs sowie des LRS-Prototyps umgesetzt werden und die von der Spezifikation der Experience API geforderte Form der Daten kann mit den gewählten Techniken realisiert werden.

Geräte- und Systemunabhängigkeit ✓

Bei dem verwendeten LernBar-WBT handelt es sich um ein speziell für mobile Geräte entwickeltes WBT, in dem auf die Verwendung von Technologien, die von mobilen Geräten nicht unterstützt werden, verzichtet wird. Das Ziel des LernBar-WBTs ist es, auf fast allen mobilen Geräten sowie Desktop-Browsern ausgeführt werden zu können. Dabei gibt es momentan noch kleinere Probleme, da sich das LernBar-WBT selbst noch in der Entwicklungsphase befindet. Die gewählten Techniken sollten allerdings dazu in der Lage sein, das WBT auf nahezu jedem Gerät und in jedem Browser ausführen zu können.

Die Implementierung der Experience API in das WBT verwendet dabei auch JavaScript, was die Ausführbarkeit in Browsern nicht weiter einschränkt. Tests haben gezeigt, dass alleine die Verfügbarkeit von CORS in den Browsern eine allgemeine Kompatibilitätsgrenze darstellt. Somit kann man sagen, dass ein WBT oder eine Webseite, welche die TinCanJS-Bibliothek zur Umsetzung der Experience API verwendet, das Ziel der Geräte- und Systemunabhängigkeit erfüllen kann.

Der erstellte LRS-Prototyp konnte unter Verwendung der Programmiersprache PHP und des slim-Frameworks auf einem Apache-Webservice implementiert werden. Da das LRS einen RESTful WebService darstellt, sollte es auch auf anderen Server-Systemen sowie in anderen Programmiersprachen umsetzbar sein.

Auf das erstellte LRS können alle Systeme und Geräte zugreifen, die http-Requests absenden können. Für Java und Objective-C existieren TinCanJS-Pendants, die in dieser Arbeit nicht analysiert wurden. Sollten sie jedoch dieselbe Funktionalität wie die TinCanJS enthalten, wäre eine verhältnismäßig einfache Implementierung der Experience API auch für native Applikationen möglich.

➔ Die Experience API kann system- und geräteunabhängig umgesetzt werden. Dies wurde beispielhaft an der prototypischen Implementierung in ein mobiles-WBT gezeigt.

Verfügbarkeit der Daten ✓

Die Implementierung des LRS-Prototyps konnte, wie oben bereits beschrieben, erfolgreich umgesetzt werden. Ein Beispiel-Reporting-Tool, der Statement Viewer, wurde bereitgestellt, um die empfangenen Statements von überall abrufen und betrachten zu können. Einschränkungen zur Verfügbarkeit der Daten durch eine Abfrage der Autorisation des Nutzers wurden nicht implementiert, die Möglichkeiten zur Umsetzung wurden jedoch besprochen.

Auch der erstellte xAPI-WBT-Prototyp kann die generierten Statements ohne Einschränkungen senden und empfangen, sofern der Browser CORS unterstützt. Szenarien für unstabile Internetverbindungen wurden ebenso bedacht und durch Nutzung des LocalStorage des Browser behandelt.

➔ Durch erfolgreiche Implementierung des LRS-Prototyps als RESTful WebService konnte gezeigt werden, dass die im LRS gespeicherten Daten von überall aus angefordert und verarbeitet werden können.

Zusammenfassend kann somit festgestellt werden, dass eine Implementierung der Experience API, die alle Anforderungen an zeitgemäße eLearning-Inhalte erfüllt, mit aktuellen technischen Mitteln möglich ist. Damit konnte nach der Analyse der Spezifikation auch die Umsetzbarkeit der Experience API gezeigt werden.

7 Fazit

Durch die Analyse und prototypische Implementierung der Experience API konnte gezeigt werden, dass die Experience API eine geeignete Technologie zur Umsetzung aktueller Anforderungen des eLearnings darstellt.

Aus der in Kapitel 4 durchgeführten Analyse von aktuellen Trends im eLearning konnten konkrete Anforderungen an ein zeitgemäßes eLearning aufgestellt werden. So sollten aktuelle Konzepte und Techniken des eLearnings Lernerfahrungen auf *mobilen Geräten*, die Vernetzung der Lernenden über *soziale Netzwerke* sowie *Learning Analytics* möglich machen, um den Anforderungen eines zeitgemäßen eLearnings gerecht zu werden.

Der aktuell verbreitetste eLearning-Standard SCORM ist nicht in der Lage diesen Forderungen nachzukommen. Die in der Entwicklung befindliche *Training and Learning Architecture* hat sich jedoch zum Ziel gesetzt, ein neues Modell bereitzustellen, das es Lehrenden ermöglicht, Inhalte zu erstellen, die diese Anforderungen erfüllen. Technische Basis und somit Kernkomponente der TLA ist die *Experience API*, deren technische Spezifikation in Kapitel 5 untersucht wurde.

Die Analyse der technischen Spezifikation der Experience API hat gezeigt, dass die Experience API eine wohldefinierte und gut durchdachte Technologie beschreibt, mit der Aktivitäten und Erfahrungen von Lernenden *system- und geräteübergreifend* erfasst sowie wieder *abgerufen* und verarbeitet werden können. Außerdem führt die sehr *offene Form der erfassten Daten* dazu, dass fast alle Aktivitäten von Lernenden abgebildet werden können, wobei es keine Rolle spielt, ob es sich um Aktivitäten im Internet oder im realen Leben handelt.

Die technische Konzeption des LRS als RESTful WebService und der Statements als JSON-Objekte sowie das integrierte Sicherheitskonzept trugen zur positiven Einschätzung der Spezifikation der Experience API bei. So blieb die Analyse der technischen Spezifikation fast ohne Kritik. Lediglich das Fehlen von Regeln für das Szenario einer temporären Offline-Phase des Clients, die fehlenden Spezifikationen zum Content Packaging und die mögliche Fragmentierung der, in Statements verwendeten Verben, wurden bemängelt. Allerdings ist durch die starke Einbindung der eLearning-Community in die Entwicklung der Experience API und der Tatsache, dass die Experience API nur ein Teil der TLA ist, davon auszugehen, dass die angesprochenen Punkte in späteren Versionen der Experience API oder in anderen Komponenten der TLA behandelt werden.

Zusätzlich zur theoretischen Analyse, der technischen Spezifikation der Experience API, wurde auch die Implementierbarkeit der Experience API unter Verwendung von aktuellen Web-Techniken in Kapitel 6 untersucht. Um eine Umsetzbarkeit der Hauptfunktionen der Client- und Server-Spezifikationen der Experience API zu

untersuchen, wurde eine prototypische Implementierung der Experience API in einem mobilen WBT durchgeführt, sowie ein einfacher Learning Record Store entwickelt.

Da für die prototypische Implementierung im mobilen WBT nur JavaScript verwendet wurde, kann die Umsetzung stellvertretend für alle Web-Applikationen interpretiert werden (Webseiten, hybride Apps, WebApps).

Somit verstärkt das Ergebnis der Implementierungen die positive Bewertung der Experience API als geeignete Technologie für aktuelle Entwicklungen im eLearning.

Die TLA könnte auf Basis der Experience API eine, allen Anforderungen des aktuellen eLearnings entsprechende, Architektur sein, die Lehrende in naher Zukunft beim Erstellen von zeitgemäßen eLearning-Inhalten unterstützt.

Wie allerdings die Entwicklung der TLA weitergehen wird, kann im Moment noch nicht definitiv bestimmt werden. So fehlen noch Spezifikationen für drei der vier Komponenten der TLA. Jedoch lässt die gute Entwicklung der Experience API darauf hoffen, dass auch diese Komponenten zielführend umgesetzt werden.

Doch auch ohne den Kontext der TLA stellt die Experience API eine gelungene Technologie dar, die aktuell in vielen Systemen Anwendung finden könnte. So können, wie gezeigt wurde, WebApps und hybride Apps sowie Webseiten die Experience API integrieren. Jedoch auch native Apps, Serious Games oder Simulationen könnten von ihr Gebrauch machen, da die benötigten Techniken systemübergreifend verfügbar sind. Die Entwickler von eLearning-Inhalten müssten Lehrinhalte nicht mehr primär für den Browser und die Integration in LMS entwickeln, sondern können alle Funktionen der Geräte nutzen und zur Steigerung der Lernerfahrung verwenden. Ebenso ist es durch die Experience API möglich, die protokollierten Daten wiederzuverwenden. Besonders in den Bereichen Learning Analytics und lebenslanges Lernen könnten somit neue Möglichkeiten geschaffen werden.

Zum Abschluss der Untersuchung der Experience API kann somit festgestellt werden, dass die Experience API fast alle in sie gelegten Erwartungen erfüllen kann. Einzig die Aussage der ADL *„Make it simple, Make it powerful“*[1] muss ein wenig korrigiert werden:

„Make it powerful and as simple as possible“

beschreibt die Experience API besser. Die Umsetzung der Experience API ist nicht simpel, aber für die vielen Möglichkeiten, die sie bietet, nicht unnötig kompliziert.

[1] Siehe 5.6.2

8 Ausblick

Abschließend sollen nun einige allgemeine Überlegungen darüber, wie sich die Experience API und die Training and Learning Architecture in naher Zukunft entwickeln könnten und welche Auswirkungen dies auf einzelne Bestandteile des eLearnings haben könnte, angestellt werden.

Ausblick - Training and Learning Architecture

Die Entwicklung der Experience API stellt erst die erste Phase der *Training and Learning Architecture* der ADL dar. Die Komponenten *Learner Profiles*, *Content Brokering* sowie die *Competency Networks*[1] besitzen noch keine festen Spezifikationen und wurden vorerst nur in ihren Ansätzen definiert. In diesen Bereichen soll zuerst geforscht, beziehungsweise abgewartet werden, welche Lösungen in der Community und von Anbietern von eLearning Inhalten geschaffen werden. Nach diesen soll sich dann die weitere Entwicklung der TLA ausrichten.

SCORM war in erster Linie für das US-Militär entwickelt worden. Laut der ADL soll mit der Entwicklung der TLA ein eher serviceorientierter Ansatz verfolgt werden.[2] Die dadurch gewählte, offene Entwicklungsstrategie, wie sie schon im Project TinCan und auch während der Spezifikation der Experience API durchgeführt wird, soll sicherstellen, dass die entstandenen Spezifikationen die Interessen der Lehrenden und WBT-Autoren widerspiegeln.

Auch die Version 1.0 der Experience API Spezifikation bildet nur eine Momentaufnahme aus den laufenden Entwicklungen[3] ab. Durch die offene Arbeit an der Spezifikation, gemeinsam mit der Community und Firmen, kann sich die Experience API noch wandeln. Sollte sich in der Zusammenführung der drei restlichen Komponenten der TLA herausstellen, dass einzelne Bereiche der Experience API angepasst werden müssen, sind Änderungen und Erweiterungen der Spezifikation denkbar.

Die Experience API stellt, wie in dieser Arbeit gezeigt wurde, eine geeignete technische Basis für einen neuen eLearning-Standard dar. Viele Projekte, die mit SCORM unmöglich waren, können mit ihr umgesetzt werden. Das volle Potenzial der TLA wird sich allerdings erst zeigen, wenn die restlichen Komponenten genauer spezifiziert wurden.

Es bleibt zu hoffen, dass die Entwicklung der weiteren Komponenten nicht erneut drei Jahre Entwicklungszeit benötigt, wie es bei der Experience API der Fall war.

Auch im Bereich der Forschung zeigt sich die Aktualität der Entwicklung der TLA und der Experience API. So existierte zu Beginn der Recherche zu dieser Arbeit keine deutschsprachige Veröffentlichung zu diesen Themen. Erst kurz vor Ende der Arbeit, im

[1] Wie in Kapitel *4.2.4 - Die Training and Learning Architecture (TLA)* vorgestellt
[2] Siehe http://www.adlnet.gov/wp-content/uploads/2013/02/TLAWebinarFeb2013-1.pdf
[3] Siehe http://www.adlnet.gov/experience-api-version-1-0-0-released

August 2013, wurde in [Gl13] die Verwendung der TLA zur Umsetzung einer mobilen App beschrieben.

Ausblick - Experience API

Durch Verwendung der Experience API werden Autoren neue Möglichkeiten in der Gestaltung und Konzeption von Lernprozessen ermöglicht. Inwieweit dies bestehende eLearning-Inhalte beeinflussen könnte, soll nun kurz besprochen werden.

Neue Arten von Lernapplikationen

Über Jahre hinweg definierte SCORM als am meisten verbreiteter eLearning Standard maßgeblich die Umsetzungsform von Lerninhalten. Zur Zeit, als SCORM entwickelt wurde, verlagerten sich viele Lebensbereiche ins Internet und definierten damit flexible Lerninhalte als allzeit erreichbare Inhalte im Internet. Lerninhalte wurden in Form von WBTs implementiert, um den Anforderungen des damaligen eLearnings gerecht zu werden.

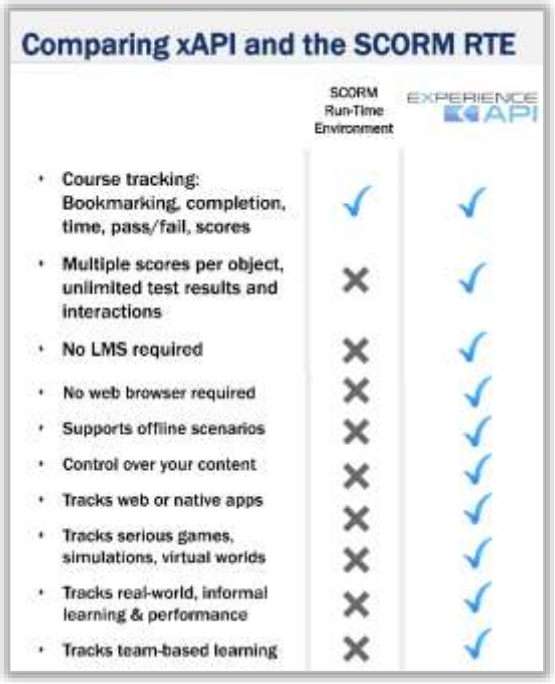


Abbildung 29 - Unterschiede SCORM <-> xAPI

Die Definition von Flexibilität hat sich heutzutage durch Einführung leistungsstarker mobiler Geräte jedoch gewandelt. Bei ihnen ist es, durch die noch nicht flächendeckend verbreitete Verfügbarkeit von mobilen Internetzugängen, ein Hindernis, wenn Inhalte nur im Web verfügbar sind. Einen flexiblen Lerninhalt definiert man aktuell als einen Inhalt, den man herunterladen, mitnehmen und ausführen kann, ohne auf eine Internetverbindung angewiesen sein zu müssen.

Eine Rückkehr zu nativen Applikationen könnte somit zu einem neuen Trend in der Entwicklung von Lerninhalten werden. Dabei sollte man allerdings nicht von einer Umkehrung der Entwicklung und einer Rückkehr zu klassischen CBTs sprechen, sondern vielmehr die Definition der WBTs erweitern. Die meisten aktuellen, mobilen, nativen Apps bieten dem Nutzer Funktionalitäten, die die bereitgestellten Inhalte mit Web-Technologien ergänzen und anreichern. Demzufolge sollte in diesem Kontext von einer neuen Generation von WBTs gesprochen werden. Diese haben zum Ziel, die Erfahrungen, die bei der Entwicklung von WBTs gesammelt wurden, auch lokal einsetzbar zu machen. Ebenso soll dabei auf die Verwendung eines Browser, der in klassischen WBTs nur ein Hilfsmittel zur Umsetzung war, verzichtet werden können.

Genau dies stellt auch die Anforderungen an die Entwicklung der Experience API dar. Sie soll online die bestmögliche Datenerfassung möglich machen, sich dabei allerdings nicht auf den Browser als Plattform beschränken. Dies wurde durch die Spezifikation der Experience API erreicht, und so wird es interessant sein zu beobachten, wie sich Lerninhalte in naher Zukunft entwickeln werden, und welche neuen Formen von Lerninhalten daraus entstehen können.

Auswirkung der Experience API auf LMS

Die Experience API könnte einen massiven Einfluss auf Learning Management Systeme, wie sie momentan verfügbar sind, haben.

Durch die Definition des LRS, dass dieses eine Nutzer- sowie Rechteverwaltung bieten muss[4], ist noch nicht abzusehen, wo sich LMS in zukünftigen Konzepten des eLearnings positionieren werden, und welche Funktionen sie bereitstellen sollten.

Somit wird es spannend zu beobachten, wie die LMS auf die vermeintliche Konkurrenz der LRS reagieren werden. Im besten Fall entsteht eine gute Zusammenarbeit der beiden Komponenten. Das LMS würde seiner Hauptfunktion als Plattform zum Kurs-Management weiterhin nachkommen. Die Datenspeicherung und -verwaltung könnte in ein externes LRS ausgelagert oder in einem, im LMS integrierten LRS, durchgeführt werden. Ebenso könnten LMS einige ihrer aktuell bereitgestellten Aktivitätsmodule wie Foren, Wikis und Chats für Lernende an externe Plattformen freigeben. Diese könnten ihre Ergebnisse dann über die Experience API miteinander austauschen. Dabei gilt es zunächst abzuwarten, in welchem Umfang sich die Experience API verbreiten wird, und wie stark ihr Einfluss auf bestehende eLearning-Konzepte wirklich sein wird.

Verbreitung der Experience API

Die Experience API wird seit ihrer Konzeptionsphase im Project TinCan in Zusammenarbeit mit vielen Firmen, die Lösungen für Umsetzungen im Bereich eLearning anbieten, entwickelt.

So konnten schon in Version 0.9 der Spezifikation Firmen gefunden werden, die die Experience API teilweise in ihre Produkte (LRS und WBTs) integriert haben. Dies zeigt einen deutlichen Bedarf nach den Möglichkeiten, die die Experience API eLearning-Inhalten bietet.

Weiterhin stellt die Community-basierte Entwicklung der Experience API sowie der durchaus hohe Werbeaufwand, den die ADL und Rustici Software zur Steigerung der Popularität der Experience API betreiben, weitere Gründe dar, aus denen sich die Experience API bald in vielen eLearning-Inhalten wiederfinden könnte. Dazu zählen Learning Management Systeme, die eine Kommunikation mit Learning Record Stores

[4] Siehe 5.3.1.10 - Authority

erlauben oder sogar ein eigenes LRS implementieren, WBTs mit Experience API Unterstützung sowie separat erhältliche LRS-Server, die die Autoren nutzen können.

Wie erfolgreich die Experience API jedoch in Produkten außerhalb des Bereichs der eLearning-Software verbreitet werden kann, hängt hauptsächlich davon ab, welche Anbieter die Experience API in ihre bestehenden Dienste integrieren.

Der Wunsch nach einer Erfassung von Aktivitäten auf Facebook, Google+, Wikipedia oder Twitter ist mit der Experience API und ihrer offenen Form der Statements zwar umsetzbar, jedoch ist es fraglich, ob diese „Global Player" im Bereich der sozialen Medien die Experience API implementieren werden.

Zum Beispiel können Beiträge auf Twitter im Moment nachträglich, mit einer Art Bookmarklet[5], das dem User zur Verfügung gestellt wird, im LRS protokolliert werden. Eine echte native Implementierung, sodass eine Information über erstellte Beiträge in Form eines Statements direkt von der Twitter-Plattform an ein LRS gesendet wird, ist allerdings unwahrscheinlich, da soziale Plattformen erfahrungsgemäß sparsam mit der (kostenlosen) Freigabe von Daten umgehen.

Sollte es der ADL jedoch gelingen, einen der großen Plattform-Anbieter zu einer Zusammenarbeit zu bewegen und die Experience API in ihre Plattform zu integrieren, wäre ein Durchbruch in der Verbreitung der Experience API möglich.

Datenschutz

Eine weitere Frage, die sich bei zunehmender Verbreitung der Experience API stellen wird, ist, ob die Daten, die im LRS gespeichert werden, überhaupt protokolliert werden dürfen.

Das Thema Datenschutz wird aktuell scharf diskutiert und auch die Experience API wird sich dieser Diskussion stellen müssen.

- Was sollen oder müssen Lehrende über ihre Studenten wissen?
- Wer sollte über die Verteilung der Daten entscheiden?
- Muss der Lernende vor jedem Aufruf eines WBTs darüber informiert werden, welche Daten übertragen werden?
- Wer stellt sicher, dass nur diese Daten übertragen werden?

Eine vorstellbare Lösung wäre, dass der Lernende selbst ein LRS-ähnliches System zur Verfügung hätte, mit dem er über seine Daten verfügen kann und anderen Systemen den Zugriff auf die Daten explizit erlauben muss. In der Realität wird das LRS meist vom Lehrenden bereitgestellt werden, sodass eine Kontrolle, was mit den gesammelten Daten wirklich passiert, schwierig wird.

[5] Wie in 6.2.4 beschrieben

Hier könnten die *Learner Profiles*, die zur TLA gehören, eine Lösung bieten. Wie diese umgesetzt werden sollen, ist noch offen und somit muss die Umsetzung des Datenschutzes in der Experience API aktuell kritisch betrachtet werden.

Verwendung der Experience API

Die Frage, ob die Experience API in aktuellen Entwicklungen eingesetzt werden sollte, kann momentan nicht eindeutig beantwortet werden.

Wie bereits erwähnt, stehen die Chancen, dass die Experience API und die TLA zu neuen Standards im eLearning werden, sehr gut. Allerdings wird an der Spezifikation der Experience API immer noch gearbeitet und so könnten Änderungen diejenigen, die die Experience API jetzt schon einsetzen, zu ständigen Updates der entwickelten Produkte zwingen.

Nichtsdestotrotz stellt die Experience API, durch die Wahl eines RESTful WebService als Basis für ein LRS, eine gute und moderne Technik für Inhalte dar, die Wert auf eine mobile Ausführbarkeit legen oder eigene Funktionen auf Basis der Experience API integrieren möchten.

Die Entscheidung, zum jetzigen Zeitpunkt auf die Experience API zu setzen, ist mit Risiken behaftet. Die Experience API bietet viele Möglichkeiten, WBTs für Autoren und Nutzer mit hilfreichen Features und Funktionen auszustatten. Ob man allerdings den Pflegeaufwand der implementierten Komponenten der Experience API in Kauf nehmen will, oder ob man besser ein Produkt entwickelt, das die Experience API verwendet, um neue Features möglich zu machen, aber dabei vorerst nicht vollständig regelkonform umsetzt, bleibt eine Entscheidung der Entwickler.

Entwickler sollten sich demzufolge bewusst sein, dass sie sich, sobald sie mit der Behauptung werben, dass das WBT xAPI-kompatibel sei, dem ständigen Druck nach Updates aussetzen, sollte sich die Spezifikation noch ändern.

Wird allerdings die Experience API dazu verwendet, das eigene WBT sowie eine Webplattform mit zusätzlichen Features für Autoren und Lernende auszustatten, so könnte man die Experience API Spezifikation soweit umsetzen, wie es für das System Sinn macht. Zum Beispiel indem man den Autoren vorerst nicht die Möglichkeit bietet, das LRS wechseln zu können. Dadurch würde nur wenig Arbeit investiert werden müssen, sollte die Experience API einmal standardisiert und final erscheinen.

Die Spezifikationen der Experience API sollen, voraussichtlich noch 2013, einem Standardisierungsgremium übergeben werden[6]. Dabei ist es bei einem Standardisierungsprozess üblich, dass die Spezifikationen überarbeitet und abgewandelt werden. Diese Standardisierung könnte der Experience API die benötige

[6] Siehe http://www.adlnet.gov/the-road-ahead-for-experience-api

Festigung bieten, die sie bräuchte, um Entwicklern letzte Zweifel an der Beständigkeit der Spezifikation zu nehmen.

Zusammenfassend kann man empfehlen, aktuelle Neuentwicklungen auf Basis der Technik der Experience API umzusetzen. Dadurch könnte von den möglichen Features profitiert und gleichzeitig die Möglichkeit offen gehalten werden, den Standard eines Tages komplett zu unterstützen.

Somit bleibt festzustellen, dass die Entwicklung der Experience API und der TLA mit dem Erscheinen von Version 1.0 der Experience API Spezifikationen keinesfalls abgeschlossen ist. Die technische Basis der Experience API ist allerdings vielversprechend und die Wahrscheinlichkeit ist hoch, dass die TLA in Kombination mit der Experience API zu einem bedeutendem eLearning-Standard werden kann. Ob die Experience API das eLearning wirklich revolutionieren kann, bleibt aber abzuwarten.

I. Literaturverzeichnis

[Ad01] Advanced Distributed Learning Initiative: SCORM 1.2 Specification. http://www.adlnet.gov/wp-content/uploads/2011/07/SCORM_1_2_pdf.zip, 02.09.2013.

[Ad03] Advanced Distributed Learning Initiative: Cross-Domain Scripting Issue. http://www.adlnet.gov/wp-content/uploads/2011/07/ADL_CrossDomainScripting_1_0.pdf, 12.08.2013.

[Ad09] Advanced Distributed Learning Initiative: SCORM 2004 4th Edition Specification. http://www.adlnet.gov/wp-content/uploads/2011/07/SCORM_2004_4ED_v1_1_Doc_Suite.zip, 02.09.2013.

[Ad12a] Advanced Distributed Learning Initiative: Experience API Companion Document. http://tincanapi.wikispaces.com/ADL+Experience+API+Companion+Document, 26.08.2013.

[Ad12b] Advanced Distributed Learning Initiative: The Experience API: Origin and Capabilities. http://www.adlnet.gov/wp-content/uploads/2012/10/Experience-API-Webinar-v2.1-Final-hw.pdf, 11.08.2013.

[Ad12c] Advanced Distributed Learning Initiative: The Unity-SCORM Integration Toolkit Version 1.0 Beta. http://www.adlnet.gov/scorm-unity-integration, 18.08.2013.

[Ad12d] Advanced Distributed Learning Initiative: An ADL Perspective on Next Generation SCORM Requirements as Derived from Project Tin Can. http://www.adlnet.gov/wp-content/uploads/2012/01/NEXTGEN-SCORM-requirements-20120130_v1.pdf, 18.08.2013.

[Ad13] Advanced Distributed Learning Initiative: Experience API - Specification. Version 1.0.0. http://www.adlnet.gov/wp-content/uploads/2013/04/20130426_xAPI_v1.0.0-FINAL.pdf, 11.08.2013.

[AI93] AICC -Aviation Industry CBT Committee: CMI - Guidelines for Interoperability. http://www.aicc.org/docs/tech/cmi001v4.pdf, 12.08.2013.

[Ba12] Banker, K.: MongoDB in action. Manning, Shelter Island, NY, 2012.

[Be12] Bekier, A.: Web Based Trainings auf mobilen Endgeräten, 2012.

[Br12] Bremer, C.: New format for online courses: the open course Future of Learning. Proceedings of the 5th International eLBA Science Conference. http://www.bremer.cx/paper49/Artikel_elba2012_opco_bremer.pdf, 02.09.2013.

[Ch13] Chodorow, K. a.: MongoDB. The definitive guide. O'Reilly Media, Sebastopol, 2013.

[Co04] Corinne Montandon: Standardisierung im e-Learning. Eine empirische Untersuchung an Schweizer Hochschulen. http://www.iwi.unibe.ch/content/publikationen/arbeitsberichte/2004/e6050/e6133/e7162/e7164/e7170/AB161.pdf, 12.08.2013.

[Co06] Corinne Montandon: Adoption von Standardisierung im e-Learning - Eine Umfrage bei e-Learning-Projekten an Hochschulen im deutschen Sprachraum. http://www.iwi.unibe.ch/content/publikationen/arbeitsberichte/2006/e6050/e6133/e6605/e6607/e7554/AB180.pdf, 12.08.2013.

[Cr06] Crockford, D.: The application/json Media Type for JavaScript Object Notation (JSON) - RFC 4627. http://tools.ietf.org/html/rfc4627, 02.09.2013.

[Cr08] Crockford, D.: JavaScript. The Good Parts. O'Reilly Media, Inc., Sebastopol, 2008.

[ED11] EDUCAUSE Learning Initiative: 7 Things You Should Know About First-Generation Learning Analytics. http://net.educause.edu/ir/library/pdf/ELI7079.pdf, 24.08.2013.

[Eh11] Ehlers, U.-D.: Qualität im E-Learning aus Lernersicht. VS Verlag für Sozialwissenschaften / Springer Fachmedien Wiesbaden GmbH, Wiesbaden, Wiesbaden, 2011.

[Gl13] Glahn, C.: LMS Integration von Microlearning Apps mit Hilfe der ADL TLA am Beispiel der Mobler Cards App Zusammenfassung. In: Claudia Bremer, Detlef Krömker(Hrsg.): E-Learning zwischen Vision und Alltag. Tagungsband der GMW-Tagung vom 02.-05. September 2013 an der Goethe Universität Frankfurt. Waxmann, S. 374-379.

[IE02] IEEE - Learning Object Metadata (LOM) working group: Draft standard for learning object metadata. (IEEE P1484.121.1). http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf, 12.08.2013.

[IM07] IMS Global Learning Consortium: Content Packaging. v1.2 Public Draft 2. http://www.imsglobal.org/content/packaging/index.html, 12.08.2013.

[Ja05] Jan Gellweiler: Die ADL SCORM Spezifikation und ihre Verbreitung. http://www.kaderali.de/fileadmin/praesentationen/gellweiler_0405_SCORM.pdf, 12.08.2013.

[LE08] LETSI: The LETSI SCORM 2.0 White Papers - SCORM - . http://scorm.com/tincanoverview/the-letsi-scorm-2-0-white-papers/, 11.08.2013.

[Ma12a] Maske, P.: Mobile Applikationen 1. Interdisziplinäre Entwicklung am Beispiel des Mobile Learning. Gabler Verlag, Wiesbaden, 2012a.

[Ma12b] Maske, P.: Mobile Applikationen 2. Interdisziplinäre Entwicklung am Beispiel des Mobile Learning. Gabler Verlag, Wiesbaden, 2012b.

[Ma13] Mathias Schäfer: Einführung in JavaScript. http://molily.de/js/, 02.09.2013.

[Mi12] Michael Dazer: RESTful APIs - Eine Übersicht. http://www.snet.tu-berlin.de/fileadmin/fg220/courses/WS1112/snet-project/restful-apis_dazer.pdf, 28.08.2013.

[MO03] MOBIlearn: Guidelines for learning in a mobile environment. http://www.mobilearn.org/download/results/guidelines.pdf, 13.08.2013.

[Pi10] Pilgrim, M.: HTML5. Up and running. O'Reilly, Sebastopol, CA, 2010.

[RR07] Richardson, L.; Ruby, S.: RESTful web services. O'Reilly, Farnham, 2007.

[Ru12a] Rustici Software: Tin Can Client Library Guidelines. http://scorm.com/wp-content/assets/tincandocs/TinCanClientLibraryGuidelines.pdf, 11.08.2013.

[Ru12b] Rustici Software: TinCan - QuickStart. http://scorm.com/wp-content/assets/tincandocs/TinCanQuickStart.pdf, 26.08.2013.

[Ru12c] Rustici Software: Tin Can API: Statements 101. http://tincanapi.com/statements-101/, 26.08.2013.

[Ru13] Rustici Software: SCORM - Run-Time Reference. http://scorm.com/scorm-explained/technical-scorm/run-time/run-time-reference/, 12.08.2013.

[Th12] The New Media Consortium: NMC Horizon Report - 2012 - Higher Ed Edition. http://www.nmc.org/pdf/2012-horizon-report-HE.pdf, 11.08.2013.

[Th13] The New Media Consortium: NMC Horizon Report - 2012 - Higher Ed Edition. http://www.nmc.org/pdf/2013-horizon-report-HE-DE.pdf, 11.08.2013.

[Ti12] Tin Can API Working Group: Experience API (aka "Tin Can API"). Tin Can API Working Group Meeting Minutes. http://tincanapi.wikispaces.com/Minutes+-+May+17%2C+2012, 30.08.2013.

II. Abbildungsverzeichnis