Eine Rahmenarchitektur für verteilte Lehr- und Lernsysteme

Matthias Gehrke, Matthias Meyer[1] und Wilhelm Schäfer
AG Softwaretechnik
Institut für Informatik
Universität Paderborn
Warburger Straße 100
33098 Paderborn
[mgehrke | mm | wilhelm]@uni-paderborn.de

Abstract

In unserer heutigen Gesellschaft ist Bildung ein zentrales Thema. Schulen, Hochschulen und Unternehmen stehen gleichermaßen vor dem Problem, zu jeder Zeit und an jedem Ort ihre Lehrangebote vermitteln zu müssen. Hierfür sind entsprechende Infrastrukturen zu ergänzen bzw. neu aufzubauen. Webbasierte Lehr-/Lernsysteme stellen eine erste, Erfolg versprechende Antwort auf diese Anforderungen dar. Solche Systeme sind jedoch häufig monolithisch programmiert. Sie sind daher kaum erweiterbar und schwer in eine bestehende IT-Landschaft zu integrieren.
Dieses Papier entwickelt eine Rahmenarchitektur für den modularen Aufbau von Lehr-/Lernsystemen, die einfach erweiterbar, anpassbar und integrierbar sind. Ausgehend von der Idee, monolithische Programme in kleine, funktionale Komponenten zu zerlegen, werden zunächst Anforderungen aufgestellt und unter deren Berücksichtigung eine komponentenbasierte Architektur für Lehr-/Lernsysteme erarbeitet. Im Anschluss wird die Komponenten-Idee erweitert und darauf aufbauend eine dienstbasierte Rahmenarchitektur präsentiert. Für deren konkrete Umsetzung wird die Verwendung von Web Services empfohlen. Dabei werden Web Services zunächst vorgestellt und ihre Eignung für die Realisierung heraus gestellt. In einem Vergleich mit anderen Techniken wird die Empfehlung begründet.
Abschließend wird der Grundaufbau eines beispielhaften Lehr-/Lernsystems mit minimaler Funktionalität unter Einsatz der Rahmenarchitektur skizziert.

1. Einleitung

Aus- und Weiterbildung sind zentrale Themen unserer heutigen Gesellschaft. Ohne eine adäquate Bildung sind die Chancen auf dem Arbeitsmarkt verschwindend gering. Wichtige Grundlagen werden hier in den Schulen bzw. Hochschulen gelegt. Hier werden grundlegende Kenntnisse für den Eintritt in die Berufswelt vermittelt und die Schüler und Studenten auf das spätere Arbeitsleben vorbereitet. Jedoch nicht nur zum Berufseintritt ist eine gute Bildung von Nöten, sondern auch während des Berufslebens bedarf es ständiger Weiterbildung. Dies haben bereits viele Unternehmen erkannt [ZFU] und in den letzten Jahren das Angebot ihrer Weiterbildungsprogramme massiv erhöht. Hierzu gehören u.a. Instruktor-basiertes Lernen (im Schulungsraum), unternehmenseigene TV-Programme (Business TV), Abspielen von Live-Mitschnitten, oder interaktives Lernen am Rechner (Computer based training). Diese Weiterbildungsprogramme sind jedoch mit erheblichen Kosten verbunden. Zum einen stehen die Mitarbeiter für die tägliche Arbeit nicht zur Verfügung und zum anderen fallen Kosten für Reisen, Hotel oder Teilnahmegebühren an.

Eine etwas andere Situation herrscht dagegen an Schulen bzw. Hochschulen. Dort ist die primäre Aufgabe die Aus- und Weiterbildung und Ausfallzeiten fallen nicht ins Gewicht. Jedoch auch hier reicht der klassische Frontalunterricht, angereichert durch Gruppenarbeiten und Praktika nicht mehr aus [NBS+99]. Die Schüler und Studenten verlangen nach ergänzenden Ausbildungsmöglichkeiten, die ihnen auch außerhalb des Hörsaals, zeitunabhängig zur Verfügung stehen. Um nun sowohl die Probleme der Firmen, als auch der Schulen und Hochschulen lösen zu können bedarf es neuer IT – Infrastrukturen, die jederzeit und an jedem Ort zur Verfügung stehen und den gewünschten Bildungserfolg erzielen.

[1] Gefördert durch das Bundesministerium für Bildung und Forschung im Rahmen des Förderprogramms „Neue Medien in der Bildung – Notebook-University“

Erste Erfolg versprechende Lösungen sind als so genannte „e-learning“ – Plattformen bekannt geworden. Diese webbasierten Systeme ermöglichen das Einstellen und Abrufen von Lehrinhalten zu jeder Zeit und an jedem Ort. Solche sowohl in der Industrie, als auch an verschiedenen Universitäten entwickelten Systeme [CS02] haben jedoch den Nachteil, dass sie oftmals nur als komplette Systeme (monolithisch) programmiert sind und nur bestimmte Funktionalitäten anbieten. Dieses Angebot an Funktionalitäten kann meistens nicht ergänzt werden, da entsprechende Schnittstellen dafür nicht vorgesehen wurden. Um dies zu verbessern und neue Funktionalitäten in bestehende Systeme integrieren zu können ist es möglich, monolithische Programme in kleine, funktionale Komponenten zu zerlegen. Aus diesen Komponenten können dann, je nach Anforderungen und individuellen Wünschen, entsprechende Lehr-/Lernumgebungen zusammengestellt werden. Zuzüglich zur reinen Funktionalität liefern diese Komponenten auch die entsprechenden Webschnittstellen. Komponenten kapseln also logisch zusammenhängende Funktionen eines Systems und bieten diese über eine Webschnittstelle an.

Um ein Zusammenspiel der Komponenten zu erreichen, müssen ihre Funktionalitäten und Schnittstellen aufeinander abgestimmt sein. Die konsequente Fortführung dieses Gedanken endet in einer Rahmenarchitektur für verteilte Lehr- und Lernsysteme, welches die Entwickler der Komponenten in die Lage versetzt, neue oder bessere Komponenten so zu entwickeln, dass sie leicht in bestehende Infrastrukturen für Lehr-/Lernsysteme integriert werden können.

In jüngster Zeit spricht man bei der Entwicklung für Webanwendungen immer häufiger von so genannten Diensten. Im Gegensatz zu Komponenten stellen Dienste eigenständige kleine Programme dar, die in einem Netzwerk anderen Programmen zur Verfügung gestellt werden. Diese Dienste können zur Laufzeit lokalisiert und deren Funktionalität verwendet werden. Bei Systemen, welche aus Komponenten bestehen muss dies vor dem Betrieb (Designzeit) erfolgen. Eine solche Flexibilität, erfordert jedoch eine erheblich genauere Definition der Dienste, deren Schnittstellen und damit einer entsprechenden Rahmenarchitektur.

In diesem Papier werden zuerst allgemeine Anforderungen an eine komponentenbasierte Architektur für Lehr-/Lernsysteme beschrieben. Danach wird eine, darauf aufbauende, Rahmenarchitektur für dienstbasierte Lehr-/Lernsysteme vorgestellt. Abschließend wird der Vorschlag für eine konkrete Realisierung mit Hilfe der neuen *WebServices*-Technik, der Firmen IBM und Microsoft, vorgestellt.

2. Anforderungen an eine komponentenbasierte Rahmenarchitektur

In diesem Kapitel werden nur Anforderungen beschrieben, die sich aus der Domäne für Lehr-/Lernsysteme ergeben und nicht ohnehin für jede Softwarearchitektur gelten. Letztere Anforderungen umfassen z.B. Schichtenarchitektur inkl. Trennung der Benutzungsschnittstelle von der Anwendung, Modularität, Kapselung, Skalierbarkeit, Verwendung von Standards usw.

2.1. Kombinierbarkeit von Komponenten

Um gewünschte Anforderungen erfüllen zu können werden häufig mehr als eine Komponente, also mehr als eine Funktionalität benötigt. Teilweise bedarf es der Kombination von Komponenten um den Anforderungen gerecht zu werden. Hieraus ergibt es sich, dass die Komponenten entsprechende Schnittstellen anbieten müssen, um technisch kombinierbar zu sein.

2.2. Individuelle Arbeitsumgebung

An das Erscheinungsbild eines Programms werden oft unterschiedliche Anforderungen gestellt. Jeder Benutzer oder Benutzergruppe (z. B. Arbeitsgruppe, Institut, etc.) eines Systems hätte gerne eine nach eigenen Wünschen aufgebaute, individuelle Oberfläche. Daraus ergibt sich die Notwendigkeit, dass sich die Oberflächen der unterschiedlichen Komponenten so kombinieren lassen, dass jeweils ein individuelles, aber für Benutzergruppen einheitliches Erscheinungsbild gewahrt bleibt (corporate identity).

2.3. Austauschbarkeit und Erweiterbarkeit

Der schnelle, technische Fortschritt und die damit einhergehende Entwicklung neuer Lehr-/Lernszenarien wird zur Weiterentwicklung bzw. Ersetzung existierender Komponenten und zu zusätzlichen Komponenten führen. Solche Strukturveränderungen/-erweiterungen müssen mit möglichst geringen Auswirkungen auf die existierenden Komponenten durchführbar sein. D. h. die Verfügbarkeit darf nicht eingeschränkt werden, oder andere Komponenten in ihrer Funktionsfähigkeit beeinträchtigt werden.

2.4. Integration in bestehende Infrastrukturen

Viele Lehr-/Lernumgebungen benötigen Informationen, die in bereits vorhandenen Systemen (legacy Systemen) erzeugt und gespeichert werden. Damit eine Neuprogrammierung und damit eine erhebliche Kosten- und Zeitsteigerung verhindert wird, ist oft die Integration dieser Systeme in die neue Umgebung notwendig. Dabei kann es notwendig sein, entweder nur auf die gespeicherten Daten dieser Systeme zugreifen zu wollen, oder aber deren Funktionalität nutzen zu wollen. Dies setzt eine komponentenorientierte Architektur dieser Systeme voraus, die es erlaubt, getrennt auf Daten und Funktionalität zugreifen zu können.

2.5. Einfache Bedienung

Die verschiedenen Komponenten, die zusammen ein Lehr-/Lernsystem bilden, werden von unterschiedlichen Nutzern bedient. Viele Nutzer werden kein tiefes, detailliertes Wissen im Umgang mit Webtechnologien haben. Sie haben spezielles Fachwissen in anderen Disziplinen (Mathematik, Chemie, Biologie, Maschinenbau, usw.) und wollen das Lehr-/Lernsystem nutzen, um ihr Wissen zu vermitteln. Aus diesem Grund muss die Bedienung der Komponenten äußert einfach, intuitiv und leicht verständlich sein.

3. Rahmenarchitektur für komponentenorientierte Systeme

3.1. Architekturschema

Eine typische Komponente in einer Lehr-/Lernumgebung weist softwaretechnisch eine klassische Dreischichtenarchitektur auf, analog zur Model-View-Controller-Architektur (MVC-Architektur, vgl. [GHJV95], [KP88]). Die unterste Schicht bildet die lokale Datenhaltungsschicht (Model). Darauf aufbauend folgt die Applikationsschicht (Controller). Die oberste Schicht wird von der Web-Schicht (View) gebildet. Abbildung 1 zeigt diesen Sachverhalt.

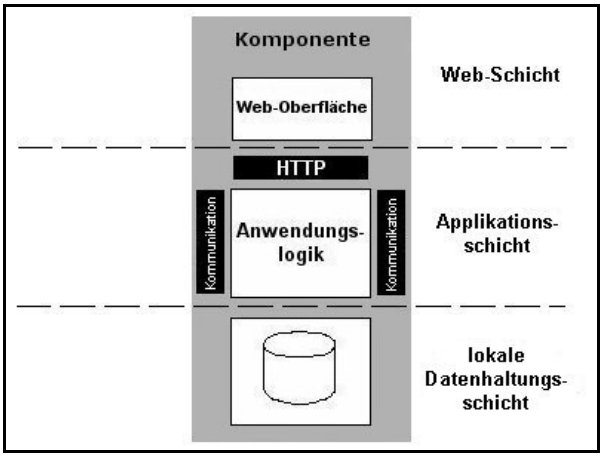

Abbildung 1: Architekturschema einer Komponente

3.2. Web-Schicht

Die Benutzungsschnittstelle (Web-Oberfläche) einer Komponente besteht aus einer Menge von untereinander verlinkten Web-Seiten. Diese werden durch einen Webclient (Internet-Browser) angezeigt, der die Seiten von dem angebundenen Webserver erhält.

Die *Adaptierbarkeit der Oberfläche* stellt eine technische Herausforderung dar. Für eine genauere Betrachtung ist eine Unterscheidung in Individualseiten, die von den Benutzern erstellt/konfiguriert werden und Systemseiten sinnvoll. Systemseiten werden von der Komponente generiert und stellen z. B. Navigationshilfen bereit, präsentieren Suchergebnisse oder bieten Funktionen an. Bei den Individualseiten handelt es sich um Seiten, die entweder komplett vom Benutzer neu erstellt wurden, oder um Systemseiten, die durch den Benutzer abgeändert werden können.

3.3. Applikationsschicht

In der Applikationsschicht befindet sich die Anwendungslogik. Diese ist bei webbasierten Systemen typischerweise vom Webclient getrennt und wird auf so genannten Webservern ausgeführt. Dort werden

Anfragen der Webclients empfangen und verarbeitet. Die Ergebnisse werden dann an den Webclient über ein entsprechendes Protokoll (HTTP[2]-Protokoll) zurückgeschickt.
Sollten für die Bearbeitung/Berechnung andere Komponenten benötigt werden, so kommunizieren diese mit definierten Kommunikationstechniken (z. B. CORBA, RPC, usw.) miteinander.

3.4. Lokale Datenhaltungsschicht

Die lokale Datenhaltungsschicht speichert z. B. persönliche Notizen der Benutzer, bei der Arbeit anfallende Zwischenergebnisse usw. Eine Speicherung in zentralen Datenbanken ist für diese Art von Daten nicht notwendig. Die Speicherung der lokalen Daten erfolgt dabei auf den Webservern und kann auf verschiedene Art und Weise erfolgen. So ist z. B. eine Speicherung in einer rel. Datenbank oder auch in Textdateien möglich.

3.5. Gesamtsystem

Wie in 3.1 beschrieben, reicht eine einzelne Komponente oftmals nicht aus und erst in der Kombination mit anderen Komponenten entsteht ein umfangreiches Lehr-/Lernsystem. Ein solches System, bestehend aus unterschiedlichen Komponenten, wird in Abbildung 2 dargestellt. Die Gesamtarchitektur unterscheidet sich in einigen Punkten von den einzelnen Komponenten.

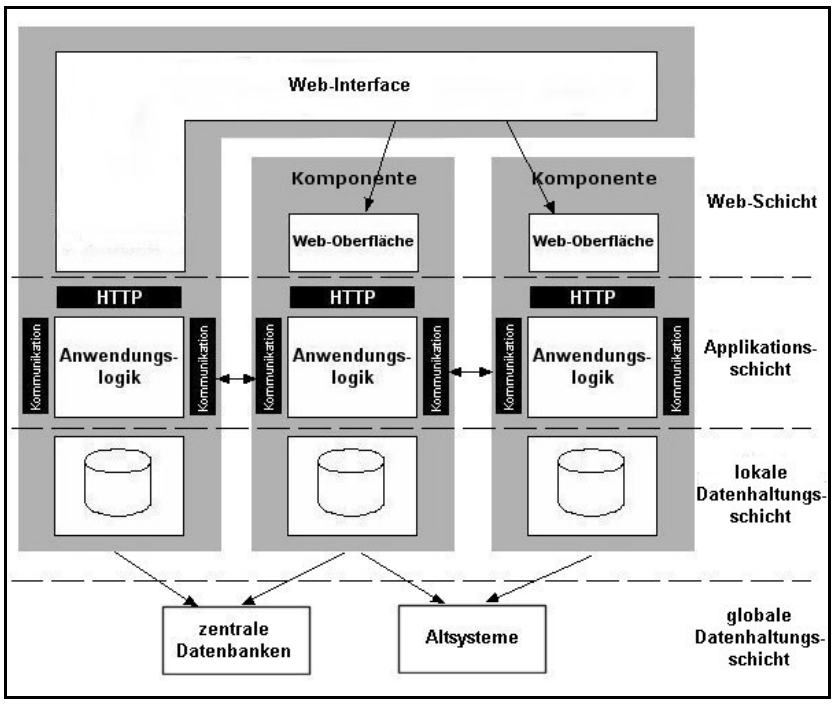

Abbildung 2: Gesamtarchitektur

3.6. Web-Interface

Im Web-Interface werden die Web-Oberflächen der einzelnen Komponenten so miteinander verknüpft, das die Gesamtoberfläche optisch als ein homogenes System erscheint.

3.7. Globale Datenhaltung

In dieser Schicht befinden sich zum einen Altsysteme (legacy Systeme), die durch definierte API's[3] von den Komponenten angesprochen werden können und zum anderen die zentralen Datenbanken in denen komponentenübergreifend Daten abgelegt sind.

3.8. Erweiterung der komponentenbasierten Idee (Dienste)

Viele der in Kapitel 2 erwähnten Anforderungen können durch ein komponentenbasiertes System erfüllt werden. Jedoch bleiben aus softwaretechnischer Sicht noch einige Probleme ungelöst. So ist beispielsweise

[2] Hypertext Transfer Protokoll
[3] Application Programming Interface

die Skalierbarkeit eines komponentenbasierten Systems sehr eingeschränkt oder die Verfügbarkeit eines solchen Systems nicht immer zu garantieren. Die Lösung solcher und weiterer Probleme wird seit einigen Jahren verstärkt durch so genannte *Dienste* adressiert.

4. Rahmenarchitektur für dienstbasierte Systeme

Die heutigen Web-Anwendungen können die an sie gestellten Anforderungen nur teilweise erfüllen. Derzeitige Systeme sind oft als klassische Client/Server Anwendungen bzw. vereinzelt als komponentenbasierte Systeme entwickelt worden. Bei diesen Systemen treten häufig jedoch Probleme wie Skalierbarkeit oder Ausfallsicherheit auf.
Diese Probleme können durch eine dienstbasierte, auf der komponentenbasierenden Idee beruhende Architektur, größtenteils umgangen werden.

Dienste erweitern die Möglichkeiten oben beschriebener Komponenten und sind durch folgende, zusätzliche Charaktereigenschaften gekennzeichnet:

- **Sie werden registriert und sind über ein Dienste Registry lokalisierbar**
  Dienste werden an einem Netz angemeldet. Dafür geben sie ihre Adresse (IP-Adresse) und ihre realisierte Funktionalität dem Netz bekannt. Zu dem Zeitpunkt, zu dem ihre Funktionalität benötigt wird, kann innerhalb des Netzes nach dieser Funktionalität gesucht und der entsprechende Dienst genutzt werden.
- **Sie unterstützen lose gekoppelte Verbindungen zwischen Systemen**
  Durch die Anmeldung am Netz wird es möglich Dienste mit gleicher Funktionalität mehrfach zu starten und am Netz anzumelden. So kann die Skalierbarkeit signifikant erhöht werden, da die Last auf mehrere, gleichartige Dienste verteilt wird. Des Weiteren wird die Ausfallsicherheit deutlich erhöht, da bei dem Ausfall eines Dienstes ein anderer Dienst mit gleicher Funktionalität einspringen kann.

Im Folgenden sollen zusätzliche Anforderungen an eine Rahmenarchitektur für dienstbasierte Systeme erläutert werden, die nicht bereits für komponentenbasierte Systeme gelten.

4.1. Kombinierbarkeit der Dienste

Im Gegensatz zu Komponenten, bei denen die Verbindungen zur Designzeit genau festgelegt werden, müssen Dienste technisch in der Lage sein zur Laufzeit andere Dienste zu suchen, sich mit ihnen verbinden und deren angebotene Funktionalität nutzen zu können. Dies erfordert eine genaue Beschreibung der Schnittstellen und eine Vorgehensweise auf welche Art und Weise bzw. in welcher Reihenfolge Dienste kombiniert werden sollen.

4.2. Sicherheit der Dienste

Um eine Arbeitsumgebung aus verschiedenen Diensten aufbauen zu können, bedarf es genau definierter Sicherheitsrichtlinien. Diese sollen verhindern, dass falsche Dienste verwendet werden und dadurch Daten verfälscht oder an unbefugte Adressaten versendet werden. Die Dienste müssen sich gegenseitig authentifizieren können um somit eine legitime Kommunikation sicherstellen zu können.

4.3. Individuelle Arbeitsumgebung

Wie schon in 2.2 erläutert, muss die Web-Oberfläche des Systems durch den jeweiligen Benutzer anpassbar sein. Bei einer dienstbasierten Architektur ist insbesondere darauf zu achten, dass jeder Dienst der eine gleichartige Funktionalität anbietet und eine Web-Oberfläche zur Verfügung stellt diese immer gleich darstellt. Jeder gleichartige Dienst muss die Einstellungen für die Web-Oberfläche zur Verfügung haben.

4.4. Verfügbarkeit und Skalierbarkeit

Obwohl es prinzipiell möglich ist einzelne Dienste (mit gleicher Funktionalität) auf verschiedenen Rechnern zu starten und damit die Verfügbarkeit zu erhöhen, kann es dennoch vorkommen, dass es keinen Dienst gibt, der eine gesuchte Funktionalität anbietet. Die dadurch entstehenden Ausfallzeiten senken in erheblichen Maße die Akzeptanz durch den Benutzer. Es muss daher sichergestellt werden, dass die verschiedenen Dienste nach einem Ausfall schnellstmöglich wieder zur Verfügung stehen. Hierfür ist ein entsprechendes Konzept zu erarbeiten.
Ein weiteres Problem stellen so genannte Stoßzeiten (Peeks) dar. Zu diesen Zeiten werden einzelne Dienste von relativ vielen Benutzern gleichzeitig genutzt. Dies kann zu erhöhten Wartezeiten führen. Dieses

Problem kann prinzipiell durch das Starten mehrerer Dienste umgangen werden. Es sollte jedoch ermittelt werden, welche Dienste häufiger als andere benutzt werden um die Anzahl der mehrfach gestarteten Dienste überschaubar (wartbar) zu halten.

4.5. Integration in die bestehende IT-Infrastruktur

Wie in 2.4 gefordert, muss es auf der einen Seite möglich sein, bestehende Systeme weiterhin verwenden zu können, auf der anderen Seite muss es die Möglichkeit geben, Daten aus diesen Systemen von Diensten auslesen und verändern zu können.
Damit dies möglich ist, müssen bestehende Systeme dahingehend angepasst werden, das sie mit Diensten kommunizieren können. Hierfür müssen entsprechende Schnittstellen spezifiziert und implementiert werden.

5. Realisierung mit Web Services

In den vorangehenden Abschnitten wird allgemein von Diensten gesprochen und es werden Anforderungen an solche Dienste formuliert. In diesem Abschnitt wird mit *Web Services* ein Konzept zur Realisierung der zuvor beschriebenen Dienste empfohlen. Im Anschluss werden Web Services zunächst vorgestellt und auf spezielle Aspekte in Bezug auf die Realisierung von Lehr-/Lernsystemen eingegangen. Darauf folgend wird Bezug zu den zuvor formulierten Anforderungen hergestellt und aufgezeigt, wie diese mit Hilfe von Web Services erfüllt werden können. Abschließend wird ein Vergleich mit anderen Techniken durchgeführt und die Empfehlung von Web Services für die Realisierung begründet.

5.1. Web Services

**„Intelligente Web Services sind für das Informationszeitalter,
was austauschbare Komponenten für das Industriezeitalter waren."**

(Scott McNealy, Chairman und CEO, Sun Microsystems, Inc.).

Seit einiger Zeit ist viel über das Konzept der Web Services zu lesen. Eine einheitliche und präzise Definition von Web Services existiert bisher allerdings nicht. So wird in [IBM00] definiert: „Web Services are self-contained, modular applications that can be described, published, located, and invoked over a network, generally, the Web.".
Die zum Zeitpunkt der Entstehung dieses Papiers wohl aktuellste Definition gibt [W3C02] mit: „A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols.".
Eine Vereinheitlichung der zahlreichen Definitionen wird in diesem Papier nicht angestrebt und ein eigener Versuch einer prägnanten Definition wird ebenso wenig unternommen. Vielmehr werden Web Services anhand ihrer Funktion und der zugrunde liegenden Standards erläutert.

Web Services sind eigenständige Applikationen, die eine bestimmte Anwendungslogik kapseln und über ein Netzwerk (Web) erreichbar sind. Unter einem Netzwerk ist in diesem Zusammenhang sowohl das Word Wide Web (WWW) als auch ein Local Area Network (LAN) zu verstehen. Um die Ansteuerung und Verwendung der gekapselten Logik zu ermöglichen, müssen Schnittstellen definiert werden. Die Definition von Schnittstellen erfolgt auf Basis der vom World Wide Web Consortium (W3C)[4] standardisierten Sprache „Web Service Description Language" (WSDL)[5]. WSDL basiert auf der „eXtensible Markup Language" (XML) und ermöglicht somit die von Plattformen und Programmiersprachen unabhängige Beschreibung der aufrufbaren Funktionen eines Web Service in einem WSDL-Dokument (vgl. [STA02]).

Web Services sowie Applikationen allgemein können mit (anderen) Web Services durch den Austausch XML-basierter Nachrichten über das vom W3C standardisierte „Simple Object Access Protocol" (SOAP)[6] kommunizieren und somit deren Logik verwenden. SOAP basiert ebenso wie WSDL auf XML und ist damit ebenfalls unabhängig von Plattformen und Programmiersprachen. SOAP ermöglicht sowohl synchrone als auch asynchrone Kommunikation, die von XML-basierten Remote Procedure Calls (RPCs)

[4] siehe http://www.w3.org
[5] siehe http://www.w3.org/TR/wsdl
[6] siehe http://www.w3.org/TR/soap12-part1

bis zum Austausch von beliebigen XML-Nachrichten reicht. Die eigentliche Nachricht wird in einem entsprechenden SOAP-Element (Body-Element) gekapselt und mit zusätzlichen Informationen (beispielsweise Routing-Informationen) in einem weiteren Element (Header-Element) versehen. Die Syntax der XML-Nachrichten, die ein Web Service erwartet und verarbeiten kann, ist in dem zum Web Service gehörenden WSDL-Dokument festgelegt (vgl. [STA02]). SOAP-Nachrichten werden über das HTTP-Protokoll versendet und empfangen (die Bindung an HTTP ist nicht zwingend notwendig, auch andere Protokolle können eingesetzt werden, vgl. [WOL01]).

Damit Web Services für andere Anwendungen lokalisierbar sind, müssen sie zum einen bei einer zentralen Instanz registriert werden und zum anderen muss diese zentrale Instanz die Suche nach bestimmten Web Services (beispielsweise auf Basis einer Schnittstellenbeschreibung) unterstützen. Zu diesem Zweck kommt der Standard „Universal Description, Discovery and Integration“ (UDDI)[7] zum Einsatz. UDDI definiert eine auf XML basierende Datenstruktur und ermöglicht die Beschreibung eines Web Service von Angaben zum Veröffentlicher über allgemeine, beschreibende Angaben zum Web Service bis hin zu dessen technischer Beschreibung, wobei häufig ein entsprechendes WSDL-Dokument (via URL) referenziert wird (vgl. [ARI00, WOL01]). Ein Verweis (URL), über den der jeweilige Web Service erreichbar ist, gehört ebenso dazu. Die Angaben werden bei einer UDDI-Registry hinterlegt und können dort entsprechend eingesehen und durchsucht werden (siehe Abbildung 3, Schritte 1, 2 und 3). Zu diesem Zweck definiert der Standard eine API, die über das SOAP-Protokoll angesteuert werden kann (vgl. [ARI00]). Unterschiedliche UDDI-Registries können die bei ihnen gemachten Eintragungen miteinander abgleichen (replizieren). Die UDDI-Regsitry wird lediglich zur Lokalisierung von Web Services benötigt. Eine spätere Kommunikation findet direkt zwischen Anwendungen und Web Services statt (siehe Abbildung 3, Schritt 4).

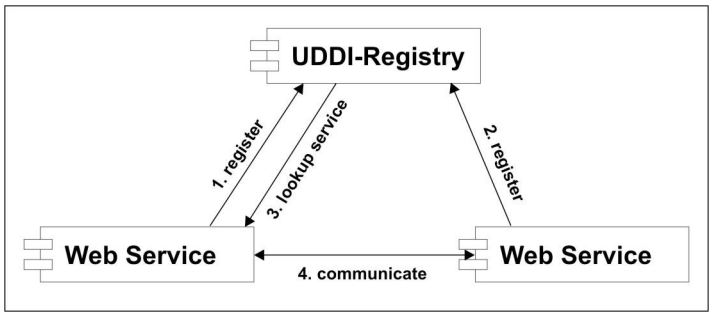

Abbildung 3: Kommunikation zwischen Web Services

5.2. Lokale Daten und Web-Oberfläche

Web Services kapseln Anwendungslogik, die häufig nicht ohne spezielle Daten ausgeführt werden kann. Web Services müssen daher in der Lage sein, anwendungsspezifische Daten zu speichern und bei Bedarf wieder abzurufen. Dabei ist eine strikte Trennung zwischen Daten, die nur für den Web Service relevant sind und Daten, die von mehreren Web Services genutzt werden, vorzunehmen. Die lokalen Daten werden von einem Web Service selbst verwaltet und sind anderen Web Services nicht zugänglich. Globale Daten hingegen werden von einem oder mehreren Web Services wieder anderen Web Services zur Verfügung gestellt. Hierbei ist darauf zu achten, dass keine Daten gleichzeitig sowohl lokal als auch global gespeichert werden (Redundanzvermeidung).

In einem Lehr-/Lernumfeld ist eine große Zahl an Benutzern zu erwarten, die zum Teil ihre eigenen Rechner mit unterschiedlichen Betriebssystemen und Softwareausstattungen mitbringen und für den Zugriff auf Lehr-/Lernsysteme verwenden. Die Bereitstellung von spezieller Client-Software für alle Systeme und Ausstattungen stellt einen erheblichen Aufwand dar. Der Aufwand vergrößert sich weiter, wenn neue Versionen der Clients herausgegeben werden und die lokal installierten Clients ausgetauscht werden müssen. Daher sollen Web Services, die bestimmte Bestandteile eines Lehr-/Lernsystems realisieren, auf die ein direkter Zugriff durch Benutzer erforderlich ist, eine HTML-basierte Benutzeroberfläche anbieten. Damit wird als Client auf Seiten der Benutzer lediglich ein HTML-Browser benötigt. Sollen über HTML hinaus weitere Techniken wie z. B. Javascript oder eingebettete Java-Applets zum Einsatz kommen, ist zu bedenken, dass dafür häufig entsprechende Browser-Plugins[8] installiert

[7] siehe http://www.uddi.org

[8] Plugins sind Hilfsprogramme, welche die Funktionalität eines Browsers erweitern.

werden müssen. Welche Techniken in welchen Versionen Verwendung finden sollen und damit von einem Browser (unter Umständen mit Hilfe entsprechender Plugins) unterstützt werden müssen, ist im konkreten Fall zu definieren.

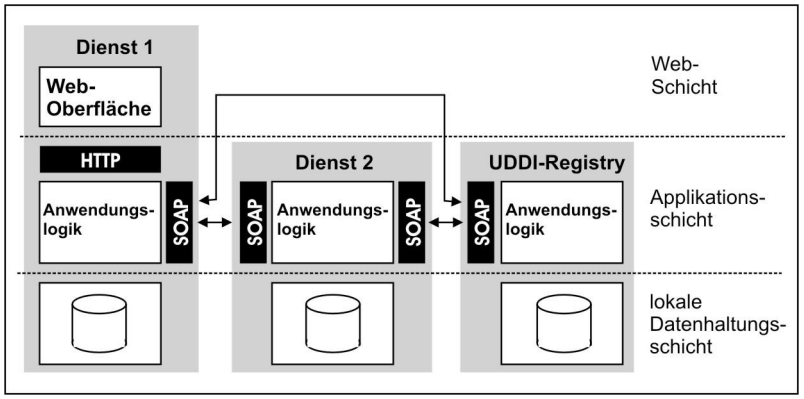

Abbildung 4: Dienstearchitektur auf Basis von Web Services

Zusammenfassend ergibt sich bei Verwendung von Web Services zur Realisierung die in Abbildung 4 gezeigte Architektur für Dienste innerhalb eines verteilten, dienstbasierten Lehr-/Lernsystems.

5.3. Verwendung von Web Services

Die unterschiedlichen Aufgaben und Funktionen von Web Services sind in WSDL-Dokumenten klar definiert. Die Web Services sind bei einer zentralen UDDI-Registry registriert und lokalisierbar. Web Services können damit andere Web Services verwenden und somit je nach Bedarf und der zu erfüllenden Aufgabe kombiniert werden. Mit Hilfe der „Business Process Execution Language for Web Services“ (BPEL4WS) besteht darüber hinaus die Möglichkeit, komplette Prozesse abzubilden, in denen verschiedene Web Services in einer festgelegten Reihenfolge aufgerufen werden und Informationen austauschen, um ein bestimmtes Endergebnis zu erreichen (vgl. [CGK+02]).

Sicherheitsaspekte wie Authentifizierung und Autorisierung können über geeignete XML-Elemente im Header der SOAP-Nachrichten abgebildet werden, die zwischen Anwendungen bzw. Web Services und Web Services ausgetauscht werden. Eine entsprechende Vorgehensweise wird in [ADH+02] mit WS-Security als SOAP-Erweiterung beschrieben. In diesem Zusammenhang erscheint die Bereitstellung eines eigenen Web Service zur Authentifizierung sowie Autorisierung sinnvoll, der es ermöglicht, dass ein Benutzer nach einmaligem Anmelden gemäß seiner Rechte auf alle Bestandteile (Web Services) des Lehr-/Lernsystems zugreifen kann (Single Sign-On).

Web Services, die Bestandteile eines Lehr-/Lernsystems kapseln, bei denen eine direkte Interaktion mit einem Benutzer stattfindet, sollen über eine HTTP-Schnittstelle eine HTML-basierte Benutzeroberfläche anbieten. Damit liegt die Gestaltung der Oberfläche in Händen der Entwickler des jeweiligen Web Service. Zur Integration der verschiedenen Oberflächen in eine gemeinsame Oberfläche ist die Bereitstellung eines speziellen Web Service denkbar, der einem Benutzer eine konfigurierbare, personalisierte Oberfläche anbietet und andere Web Services bzw. deren Benutzeroberflächen entsprechend einbindet.

Zur Definition der Schnittstellen von Web Services, für ihre Registrierung sowie für die Kommunikation mit Web Services werden plattform- sowie programmiersprachenunabhängige Standards eingesetzt. Daraus ergibt sich, dass die Implementierung der eigentlichen Anwendungslogik in einer beliebigen Programmiersprache erfolgen kann. Zudem sind die Schnittstellen klar definiert und erlauben einen transparenten Zugriff auf die gekapselte Funktionalität, ohne Kenntnisse über deren konkrete Implementierung besitzen zu müssen. Web Services sind daher einfach austauschbar. Um eine neue Funktionalität in das Lehr-/Lernsystem zu integrieren, muss lediglich ein neuer Web Service realisiert, dessen Schnittstellen in einem WSDL-Dokument definiert und bei der UDDI-Registry registriert werden. Der neue Web Service steht dann im System zur Verfügung, das so einfach erweitert werden kann.

Verschiedene UDDI-Registries können miteinander replizieren, das heißt, die bei ihnen hinterlegten Informationen bzgl. Web Services miteinander abgleichen. Somit ist es möglich, in einem Lehr-

/Lernsystem beispielsweise zwei UDDI-Registries einzusetzen, die miteinander replizieren. Eine der beiden fungiert als Backup-System und kann bei Ausfall der ersten Registry einspringen. Die Verfügbarkeit der UDDI-Registry kann damit verbessert werden.
Um die Verfügbarkeit einzelner Web Services zu erhöhen, ist die Einführung eines speziellen Monitor Web Service denkbar, der alle übrigen Web Services des Systems periodisch auf Verfügbarkeit überprüft. Erhält der Monitor nach Ablauf der Überprüfungsperiode auf eine Anfrage keine Rückmeldung, kann er beispielsweise einen Systemadministrator informieren oder selbst adäquate Maßnahmen ergreifen. Um wiederum die Verfügbarkeit des Monitors sicher zu stellen, sind weitere Mechanismen erforderlich, auf die an dieser Stelle nicht weiter eingegangen wird.
Web Services, auf die, möglicherweise zu bestimmten Zeiten, übermäßig zugegriffen wird, sind im System mehrfach zur Verfügung zu stellen, um weiterhin akzeptable Antwortzeiten zu erzielen. Prinzipiell kann dabei so vorgegangen werden, dass mehrere Web Services gleichen Typs im System gestartet werden, auf die über einen speziellen Web Service mit gleicher Schnittstelle zugegriffen wird. Dieser spezielle Web Service fungiert als Verteiler und leitet eingehende Anfragen an Web Services weiter, deren aktuelle Auslastung eine Bearbeitung zulässt. Bei Bedarf kann dieser Web Service auch einen weiteren Web Service des entsprechenden Typs starten sowie beenden. Inwieweit mit dieser Vorgehensweise Vorteile erzielt werden können, ist im konkreten Fall zu untersuchen und hängt z. B. davon ab, wie aufwendig die Ausführung der Anwendungslogik des Web Service ist oder wie viele lokale Daten benötigt werden und wie diese abgelegt sind. Denkbar ist ebenso, dass lediglich die Logik innerhalb eines Web Service mehrfach ausgelegt und der Web Service um eine zusätzliche Verteilungslogik erweitert wird.

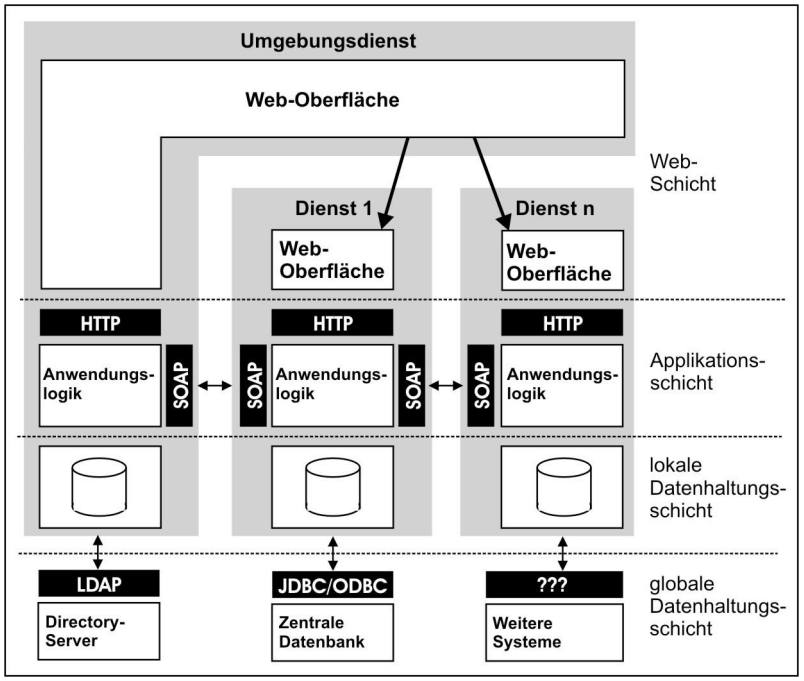

Abbildung 5: Gesamtarchitektur mit Web Services[9]

Im vorangehenden Abschnitt wird die strikte Trennung von lokalen und globalen Daten gefordert. In vielen Alt-Systemen (Legacy-Systeme) liegen Daten vor, die den Web Services zur Verfügung gestellt werden sollen (globale Daten). Damit ein Zugriff auf die in solchen Systemen gehaltenen Daten durch Web Services erfolgen kann, müssen die Systeme in das Lehr-/Lernsystem integriert werden. Dabei sind prinzipiell zwei Vorgehensweisen denkbar. Entweder die Legacy-Systeme werden als Web Services neu implementiert oder es wird ein Web Service als Adapter implementiert, der über proprietäre Schnittstellen auf die Daten zugreift und über eine eigene Schnittstelle anderen Web Services anbietet (vgl. z. B. [FWK02]).

[9] In der Abbildung ist die UDDI-Registry lediglich aus Gründen der Übersicht nicht aufgeführt. Sie muss ebenfalls (mindestens einmal) als Dienst im System zur Verfügung stehen (vgl. Abbildung 4).

Insgesamt ergibt sich bei Realisierung der Dienste mit Hilfe von Web Services die in Abbildung 5 gezeigte Gesamtarchitektur.

5.4. Vergleich mit anderen Techniken

Aus den obigen Ausführungen ergibt sich, dass mit Hilfe von Web Services die zuvor formulierten Anforderungen an dienstbasierte Lehr-/Lernsysteme erfüllt werden können. Web Services sind damit grundsätzlich zur Umsetzung solcher Systeme geeignet. Hinzu kommt, dass das von großen Unternehmen wie Microsoft und IBM entwickelte und voran getriebene Konzept der Web Services auf breites Interesse in der Industrie stößt.
Neben Web Services stehen weitere Techniken wie z. B. JINI, CORBA oder DCOM zur Verfügung, die eine Kommunikation zwischen verteilter Software ermöglichen und grundsätzlich ebenfalls zur Realisierung von Diensten eingesetzt werden können.

JINI[10] basiert auf der plattformunabhängigen Programmiersprache Java. Mit Hilfe von JINI können Dienste implementiert werden, die über einen so genannten Lookup-Service, vergleichbar mit der UDDI-Registry, lokalisierbar sind. Zur Kommunikation mit einem Dienst muss von dem in Java implementierten Lookup-Service ein serialisiertes Java-Objekt, ein so genanntes Proxy-Objekt, heruntergeladen werden, in dem die Schnittstelle des Dienstes definiert ist. Werden Methoden auf dem Proxy-Objekt aufgerufen, greift dieses auf den eigentlichen Dienst zu. Das Proxy-Objekt muss in Java implementiert sein, so dass eine Applikation, die einen JINI-Dienst benutzen will, ebenfalls in Java implementiert sein muss.
Daher ermöglicht JINI aufgrund der Plattformunabhängigkeit von Java zwar eine plattformübergreifende Dienstenutzung, jedoch ist JINI nicht unabhängig von Programmiersprachen.

Der von der Object Management Group (OMG)[11] entwickelte Standard „Common Object Request Broker Architecture“ (CORBA)[12] ermöglicht die Kommunikation mit entfernten Software-Objekten. Dabei gibt es ein Software-Objekt, das beispielsweise einen Dienst implementiert und ein Client-Objekt für den Zugriff auf den Dienst. Die Schnittstelle eines Software-Objektes wird in der OMG „Interface Definition Language“ (IDL) definiert, vergleichbar mit dem WSDL-Standard für Web Services. Auf Basis der IDL-Definition werden so genannte Client-Stubs und Object-Skeletons für verschiedene Programmiersprachen generiert. Eine Applikation, die auf ein entferntes Software-Objekt zugreifen möchte, muss den entsprechenden Client-Stub integrieren. Die Ansteuerung des entfernten Objektes erfolgt über den Client-Stub, der beispielsweise einen Methodenaufruf an einen „Object Request Broker“ (ORB) weiterleitet. Der ORB wiederum transformiert den Aufruf und leitet ihn an das entsprechende Software-Objekt über das Object-Skeleton weiter. Client-Stub und Object-Skeleton sowie Objekt-Implementierung müssen dabei nicht denselben ORB verwenden. Über das standardisierte „Internet Inter-ORB Protocol“ (IIOP) können unterschiedliche ORBs miteinander kommunizieren (vgl. [OMG02], [TVS02]).
IDL stellt eine eigenständige Sprache mit eigener, neu zu erlernender Syntax dar, während WSDL auf XML aufbaut. Die IDL-Definition kann mit Hilfe entsprechender Compiler in viele Programmiersprachen (wie z. B. C, C++, Java, COBOL, Smalltalk, Ada, Lisp oder Python) übersetzt werden, jedoch nicht in alle. Darüber hinaus muss die Generierung eines Client-Stubs für eine bestimmte Programmiersprache explizit vorgenommen werden. Web Services können auf Basis des SOAP-Protokolls generell ohne weiteres Zutun von jeder Programmiersprache aus angesteuert werden.
Ein weiteres Problem kann bei CORBA in Bezug auf die Kommunikation der Dienste zur Laufzeit entstehen. Diese Kommunikation erfolgt grundsätzlich über einen ORB. Steht dieser ORB zwischenzeitlich nicht zur Verfügung, kann auch keine Kommunikation stattfinden. Im Gegensatz dazu erfolgt die Kommunikation bei Web Services direkt zwischen Web Services bzw. Anwendungen und Web Services.

Mit dem „Distributed Component Object Model“ (DCOM) steht eine weitere Technik zur Integration von verteilten Software-Komponenten zur Verfügung. DCOM ist eine Erweiterung des „Component Object Model“ (COM)[13]. Beide Techniken wurden von der Microsoft Corporation[14] entwickelt. Software-Komponenten, mit denen via DCOM kommuniziert werden soll, müssen einem binären Format entsprechen, das durch COM spezifiziert wird. Ähnlich zu CORBA gibt es eine Microsoft IDL, basierend

[10] Siehe [SUN01a], [SUN01b].
[11] siehe http://www.omg.org
[12] siehe [OMG02].
[13] siehe [MSC95].
[14] siehe http://www.microsoft.com

auf der „Distributed Computing Environment“ (DCE) IDL, in der die Schnittstellen von Komponenten definiert werden. Zu den IDL-Definitionen werden mit Hilfe des Microsoft IDL Compilers Proxy-Objekte und Stubs erzeugt. Applikationen rufen Methoden auf dem Proxy-Objekt auf, das Proxy-Objekt gibt den Aufruf via DCE RPC an den Stub weiter, der wiederum die eigentliche Komponente ansteuert. Sowohl auf dem lokalen als auch auf dem entfernten Rechner wird eine COM-Laufzeitumgebung benötigt. Das Proxy-Objekt ist in der lokalen, der Stub in der entfernten Laufzeitumgebung angesiedelt (vgl. [COR01], [MSC95], [TVS02]).
Zur Implementierung von Komponenten und Anwendungen können alle Programmiersprachen eingesetzt werden, die das binäre Format unterstützen. Dazu zählen alle Sprachen von Microsoft, wie z. B. Visual Basic, Visual C++ oder J++. Darüber hinaus ist z. B. auch die Verwendung von Java möglich. Jedoch wird das Format nicht von allen Programmiersprachen unterstützt. Web Services sind hingegen vollkommen unabhängig von der eingesetzten Programmiersprache.
Hinzu kommt, dass DCOM zwar auf Microsoft Windows Betriebssystemen gut unterstützt wird, dies auf anderen Systemen jedoch nicht in gleichem Maße der Fall ist. „COM and DCOM are best supported on Windows 95 and NT platforms. However, Microsoft has released a version of COM/DCOM for MacOS […]. Software AG, a Microsoft partner, has released DCOM for some UNIX operating systems […] and Linux. However, DCOM over non-Windows platforms has few supporters. Until DCOM for alternate platforms has solidified, the technology is best applied in environments that are primarily Windows-based.“ [COR01]. Web Services unterliegen aufgrund der zugrunde liegenden plattformunabhängigen Standards keinerlei Einschränkungen in Bezug auf unterschiedliche Betriebssysteme. Da jedoch Web Services ebenfalls von Microsoft maßgeblich mitentwickelt werden, ist die Zukunft von DCOM nicht klar ersichtlich.

Allen erwähnten Techniken (JINI, CORBA und DCOM) ist gemein, dass sie nicht explizit für die Kommunikation über das Web entwickelt wurden, während Web Services auf standardisierte Web-Protokolle aufbauen und eine Kommunikation beispielsweise durchgängig über HTTP ermöglichen. Zwar besteht mittlerweile auch für die übrigen Techniken die Möglichkeit der Kommunikation via HTTP, jedoch nur über proprietäre Lösungen.

Aufgrund der hier identifizierten Probleme in Bezug auf die eingangs formulierten Anforderungen, die bei der Verwendung anderer Techniken auftreten können und der zuvor präsentierten Eignung von Web Services zur Erfüllung der Anforderungen propagiert dieses Papier die Verwendung von Web Services zur Realisierung verteilter, dienstbasierter Lehr-/Lernsysteme.

6. Ein beispielhaftes Lehr-/Lernsystem auf Basis von Web Services

Aus den zuvor präsentierten Anforderungen und der Architekturentscheidung ergeben sich bestimmte Dienste, die generell in einem verteilten, dienstbasierten System benötigt werden. Im Kontext des Lehr-/Lernumfelds können darüber hinaus weitere Dienste identifiziert werden, die speziell in einem Lehr-/Lernsystem existieren müssen.

Die Menge der hier aufgeführten Dienste ist keineswegs vollständig, sondern soll vielmehr als ein erster, beispielhafter Grundaufbau eines Lehr-/Lernsystems mit minimaler Funktionalität dienen. Das System ist im konkreten Fall um weitere Dienste zur Realisierung zusätzlicher Funktionalitäten zu ergänzen. Damit die Dienste des Systems lokalisierbar sind, muss zudem eine UDDI-Registry zur Verfügung stehen, bei der die verschiedenen Dienste registriert werden.

Allgemeine Dienste

Authentifizierung und Autorisierung (Security Service)

Zu den allgemeinen Diensten gehört z. B. ein Dienst, der die Authentifizierung und Autorisierung von Benutzern gegenüber dem System übernimmt (Security Service). Dieser Dienst soll über den reinen Anmeldevorgang hinaus ein Single Sign-On ermöglichen, so dass Benutzer sich nur einmalig am Gesamtsystem anmelden müssen und danach ohne eine weitere Anmeldung auf alle Teilsysteme zugreifen dürfen, zu deren Benutzung sie autorisiert sind. Der Dienst soll als Anlaufstelle für alle weiteren Dienste des Systems dienen, die bei einem Zugriff die Rechtesituation überprüfen müssen.

Benutzerverwaltung (Directory Service)

Der Security Service setzt voraus, dass eine Benutzerverwaltung mit Hilfe eines Verzeichnisdienstes verfügbar ist (Directory Service). Von diesem Dienst sind die zugriffsberechtigten Benutzer mit beispielsweise Login- und Passwortdaten zu verwalten sowie entsprechende Berechtigungen zur Nutzung bestimmter Teile des Gesamtsystems zu vergeben. Andere Mechanismen zur Authentifizierung wie z. B. digitale Signaturen oder biometrische Verfahren können ebenso eingesetzt werden.

Benutzeroberfläche (UI Service)

Darüber hinaus wird ein Dienst benötigt, der einem Benutzer den Zugriff auf das System in Form einer geeigneten Benutzeroberfläche ermöglicht (UI Service). Sinnvoll ist in diesem Zusammenhang, dass die Oberfläche personalisiert und flexibel konfigurierbar ist, so dass die Benutzeroberflächen ausgesuchter Dienste zu einer Oberfläche kombiniert und nach persönlichen Vorlieben angeordnet werden können (vgl. Abschnitt 5.3), analog zu einem personalisierbaren Portal. Ein Benutzer kann damit den Einstieg in das System gemäß seiner Vorlieben und Bedürfnisse gestalten.

Spezielle Dienste

Veranstaltungsverwaltung (Course Service)

In einem Lehr-/Lernsystem muss ein Dienst vorhanden sein, der Informationen bzgl. der verfügbaren Lerneinheiten bzw. Kurse verwaltet, vergleichbar mit einem Vorlesungsverzeichnis im Hochschulumfeld (Course Service). Zu den zu verwaltenden Informationen gehören beispielsweise der Name der Veranstaltung, von welchem Dozenten sie betreut wird, Details zu Prüfungen und welche Materialien wo zu finden sind. Lernende müssen die verwalteten Informationen anzeigen können und Lehrenden muss die Möglichkeit zur Pflege der Informationen gegeben werden. Eine Funktionalität, über die sich Lernende für die Veranstaltungen anmelden können, ist je nach konkretem Bedarf ebenfalls wünschenswert.

Über die Verwaltung von reinen Kursdaten und den Verweis auf Materialien hinaus soll der Dienst auch die Ablage bzw. den Abruf von speziell für Online-Lernen (Web-Based Training, WBT) aufbereiteten Inhalten anbieten. In diesem Zusammenhang ist es sinnvoll eine Schnittstelle anzubieten, die die Verarbeitung von Inhalten im SCORM-Format (Sharable Content Object Reference Model)[15] ermöglicht.

Verwaltung von Übungen (Exercise Service)

Zu Kursen werden in der Regel Übungen angeboten. Zu deren Verwaltung ist ein weiterer Dienst sinnvoll (Exercise Service), der auch die Anmeldung von Lernenden für Übungen sowie die Einreichung von Lösungen ermöglicht. Eine entsprechende Inhaltsanbindung beispielsweise basierend auf dem SCORM-Format ist hier ebenso zu schaffen. Lehrende müssen die Übungsinformationen pflegen können.

Prüfungsverwaltung (Exam Service)

Werden Prüfungen zu Kursen durchgeführt, muss ein Dienst realisiert werden, der Prüfungen sowie deren Ergebnisse verwaltet und ebenfalls Anmeldungen entgegennimmt (Exam Service). Prüfungen müssen über den Dienst von Lehrenden gepflegt werden können, wobei auch die Definition von Abhängigkeiten zwischen Prüfungen sinnvoll ist, so dass bestimmte Prüfungen nur dann von einem Lernenden absolviert werden dürfen, wenn dieser bereits andere Prüfungen bestanden hat. Eine Online-Durchführung der Prüfungen ist je nach Bedarf ebenfalls vorzusehen.

Digitales Studienbuch (StudyRecord Service)

Darüber hinaus muss ein Dienst bereit stehen, der einem Lernenden einen Überblick über die von ihm belegten Kurse, Übungen und Prüfungen sowie deren Ergebnisse liefert (StudyRecord Service). Ein solcher Dienst ist vergleichbar mit einem Studienbuch im Hochschulumfeld. Wünschenswert ist in diesem Zusammenhang die Integration einer Coaching-Komponente, die den Lernenden im Hinblick auf die Belegung weiterer, möglicherweise aufbauender Kurse in Abhängigkeit der bisher konsumierten Lerninhalte berät.

[15] Für weitere Informationen zu SCORM siehe z. B. [ADL01] sowie allgemein http://www.adlnet.org.

7. Zusammenfassung und Ausblick

Im vorliegenden Papier werden die Probleme und Anforderungen moderner Lehr-/Lernsysteme beschrieben. Dabei wurde gezeigt, das gängige, am Markt verfügbare Systeme jeweils nur Teile dieser Anforderungen erfüllen können. Viele solcher Systeme sind große, monolithische Programme die nur sehr schwer erweiterbar und anpassbar sind und in den seltensten Fällen Schnittstellen, für die Kommunikation mit anderen Systemen, zur Verfügung stellen. Dies müssen heutige Lehr-/Lernsysteme aber leisten um gerade im Umfeld von Universitäten und anderen großen Bildungseinrichtungen sinnvoll eingesetzt werden zu können. Als Lösung wurde die relativ neue Technik der Web Services vorgestellt. Es wurde ihre Auswahl begründet und erste Vorschläge für eine sinnvolle Aufteilung in einzelne, funktionale Programme (Dienste) vorgenommen.
Abschließend soll noch darauf hingewiesen werden, dass auch andere Techniken, die ebenfalls beschrieben werden, eine Lösung darstellen. Der Einsatz der Web Services wurde deshalb gewählt, weil sie die Anforderungen an moderne Lehr-/Lernsysteme, nach Meinung der Autoren, am besten adressieren.

8. Literatur

[ADH+02] Bob Atkinson, Giovanni Della-Libera, Satoshi Hada, Maryann Hondo, Phillip Hallam-Baker, Johannes Klein, Brian LaMacchia, Paul Leach, John Manferdelli, Hiroshi Maruyama, Anthony Nadalin, Nataraj Nagaratnam, Hemma Prafullchandra, John Shewchuk, Dan Simon. *Web Services Security (WS-Security)*, April 2002.

[ADL01] Advanced Distributed Learning Initiative. *Sharable Content Object Reference Model (SCORM). The SCORM Overview*, Version 1.2, October 2001.

[ARI00] ARIBA, INC., INTERNATIONAL BUSINESS MACHINES CORPORATION AND MICROSOFT CORPORATION. *Universal Description, Discovery and Integration (UDDI) Technical White Paper*, September 2000.

[CGK+02] Francisco Curbera, Yaron Goland, Johannes Klein, Frank Leymann, Dieter Roller, Satish Thatte, Sanjiva Weerawarana. *Business Process Execution Language for Web Services*, Version 1.0, July 2002.

[COR01] Santiago Comella-Dorda. *Component Object Model (COM), DCOM, and Related Capabilities*, Software Technology Review, Carnegie Mellon Software Engineering Institute, März 2001, http://www.sei.cmu.edu/str/descriptions/com.html am 12.11.2002.

[CS02] Initiative *CampusSource* des Landes NRW. November 2002, http://www.campusSource.de am 17.11.2002.

[FWK02] Paul Fremantle, Sanjiva Weerawarana, and Rania Khalaf. *Enterprise Services*, Communications of the ACM, Vol. 45, No. 10, ACM Press, October 2002.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*, Addison-Wesley, Reading, MA, 1995.

[IBM00] IBM Web Services Architecture Team. *Web Services architecture overview*, September 2000, http://www-106.ibm.com/developerworks/webservices/library/w-ovr/ am 21.10.2002.

[KP88] Glenn E. Krasner and Stephen T. Pope. *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*, Journal of Object-Oriented Programming, 1(3):26–49, August/September 1988.

[MSC95] Microsoft Corporation. The *Component Object Model Specification*, Draft, Redmond, Oktober 1995, http://www.microsoft.com/com/resources/comdocs.asp am 22.11.02.

[NBS+99] M. Nagl, H. Balzert, H.-W. Six, W. Schäfer, U. Kelter, A. Behle, B. Westfechtel, C. Weidauer, P. Pauen, J.Voss, J.P. Wadsack. *Studie über Softwaretechnische Anforderungen an multimediale Lehr- und Lernsysteme*, September 1999.

[OMG02] Object Management Group. *Common Object Request Broker Architecture: Core Specification*, Version 3.0, November 2002, http://www.omg.org/cgi-bin/doc?formal/02-11-03 am 22.11.02.

[STA02] Michael Stal. *Web services: beyond component-based computing*, Communications of the ACM, Vol. 45, No. 10, ACM Press, October 2002.

[SUN01a] Sun Microsystems. *Jini Architecture Specification*, Version 1.2, December 2001.

[SUN01b] Sun Microsystems. *Jini Technology Core Platform Specification*, Version 1.2, December 2001.

[TVS02] Andrew Tanenbaum und Maarten van Steen. *Distributed Systems, Principles and Paradigms*, Prentice Hall, 2002.

[W3C02] Web Services Architecture Working Group. *Web Services Architecture Requirements*, W3C Working Draft, October 2002, http://www.w3.org/TR/wsa-reqs am 21.10.2002.

[WOL01] Roger Wolter, MICROSOFT CORPORATION. *XML Web Services Basics*, December 2001.

[ZFU] Zentralstelle für Fernunterricht, Bundesinstitut für Berufsbildung. *Auszug aus dem Ratgeber für Fernunterricht der staatlichen Zentralstelle für Fernunterricht (ZFU) und dem Bundesinstitut für Berufsbildung (BIBB).* http://www.academy24.com/download/files/flg_weiterbildung.pdf