Medienzentrum der Universität
Dortmund
Emil-Figge-Str. 50
44227 Dortmund

Universität Dortmund

Medienzentrum

CampusSource Engine

Überlegungen zur Entwicklung einer modularen Lernplattform

Josef Hüvelmeyer
Arne von Irmer
Carsten Ritterskamp
Thomas Wenk

CampusSource Engine

Überlegungen zur Entwicklung einer modularen Lernplattform

Einleitung

> *"In a society which emphasizes teaching, children and students - and adults - become passive and unable to think or act for themselves. Creative, active individuals can only grow up in a society which emphasizes learning instead of teaching."*
>
> *Alexander et al. (1977): A Pattern Language*

Die universitäre Lehre hat in den letzten Jahren eine Veränderung ihres Profils erfahren: tradierte Präsenzveranstaltungen werden erweitert von Szenarien des *Blended Learning*, die Verbindung klassischer Elemente der Präsenzlehre mit webbasierten Lernplattformen birgt für die Lehre einen Zugewinn an Möglichkeiten in sich und markiert die erste Etappe auf dem Weg zu ubiquitären Lern-Netzwerken der Wissensgesellschaft.

Zwar existiert zum gegenwärtigen Zeitpunkt eine Vielzahl von Lernplattformen, diese müssen aber in der Regel als Insellösung charakterisiert werden, da sie oftmals nur für ein spezifisches Einsatzszenario konzipiert wurden, eine monolithische Softwarearchitektur aufweisen und hinsichtlich der von ihnen angebotenen Funktionalitäten nicht erweiterbar sind (vgl. Gehrke et al. (2002)). Ließen sich die geringe Kompatibilität der einzelnen Plattformen zueinander sowie ihre häufig begrenzte Erweiterbarkeit und Adaptivität in den frühen Phasen ihres Einsatzes noch verschmerzen, so erweisen sich diese Eigenschaften nun als ernstzunehmender Hemmschuh einer gleichermaßen wünschenswerten wie notwendigen Vereinheitlichung der an einer Universität vorhandenen Plattformen: eine hochgradig segmentierte Landschaft von Lernplattformen wird sich in Zukunft insbesondere vor dem Hintergrund reduzierter finanzieller Mittel kaum noch ausreichend warten, geschweige denn neuen Anforderungen gegenüber anpassen lassen – genau dies ist aber vor dem Hintergrund einer zumindest im europäischen Rahmen unaufhaltsam voranschreitenden Internationalisierung der Lehre, des Content Sharing und der kostengetriebenen Rationalisierung des Hochschulbetriebs erforderlich.

Die von den einzelnen Institutionen zu bewältigen Problemstellungen sind in wesentlichen Bereichen vergleichbar und gruppieren sich häufig um Grundfunktionalitäten einer Lernplattform herum. Exemplarisch seien die folgenden Punkte genannt:

- Personalisierter Zugang zum System und Authentifizierung der Nutzer gegenüber etablierten Nutzerdatenbanken (z.B. HIS an bundesdeutschen Hochschulen), auch mit dem Ziel eines universitätsweiten *Single Sign On.*

- Entwicklung und Austausch nach Möglichkeit standardisierter Inhalte (*Content*) für Lehre und Forschung, Anbindung externer Publikationsserver unter Verwendung standardisierter Protokolle.

- Bereitstellung von Schnittstellen zur Integration möglicherweise externer Dienste wie z.B. einer webbasierten Literaturrecherche in den Beständen der Universitätsbibliothek.

- Unterstützung von Studienkontenmodellen wie z.B. ECTS (European Credit Transfer and Accumulation System).

Über die soeben angesprochenen, standortübergreifenden Anforderungen hinaus muss eine Lernplattform stets auch Ansprüchen genügen, die sich beispielsweise aus der Einpassung der Plattform in die an einer Hochschule bereits vorhandene IT-Infrastruktur ergeben oder aus der Notwendigkeit einer Integration ausdifferenzierter Funktionalitäten zur Unterstützung hochschulspezifischer Prozesse resultieren. Vor diesem Hintergrund erscheint es notwendig, beim Entwurf einer Lernplattform gleichermaßen auf deren Erweiterbarkeit um neue Funktionalitäten wie auch auf eine weitreichende Unterstützung der anwenderspezifischen Konfiguration und (Re-)Kombination ihrer Bestandteile zu achten.

Die vorausgehend skizzierten Überlegungen und Randbedingungen legen – zunächst allgemein gesprochen – eine modulare Architektur der angestrebten Lernplattform nahe, welche neben den im weiteren Verlauf dieses Beitrags noch eingehender betrachteten (software-)technologischen Vorteilen insbesondere einen hochschulübergreifenden, kooperativen Entwicklungsprozess ermöglicht: eine kooperativ erarbeitete, verbindliche Architektur- und Schnittstellenspezifikation als Grundlage vorausgesetzt, lassen sich sowohl Basisfunktionalitäten als auch strandortspezifische Komponenten voneinander entkoppelt und weitestgehend parallel entwickeln.

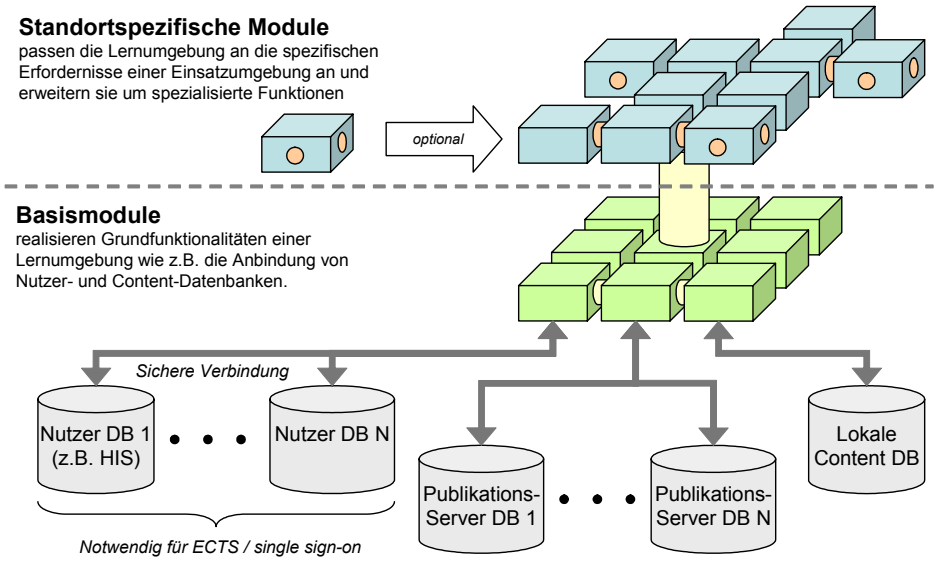

Abbildung 1: Aufgaben von Basismodulen und standortspezifischen Erweiterungen

Abbildung 1 verdeutlicht den modularen Aufbau einer Lernplattform. Jedes Modul stellt für das Gesamtsystem eine funktional geschlossene Einheit dar und verfügt über eine genormte Schnittstelle, welche die Verbindung des Moduls mit anderen Bestandteilen des Systems (i.d.R. mit weiteren Modulen) ermöglicht. Gemäß den vorausgehenden Ausführungen werden Basismodule von standortspezifischen Modulen unterschieden: erstere stellen „vitale“ Grundfunktionalitäten einer Lernumgebung wie z.B. Authentifizierungs- und Autorisierungsmechanismen zur Verfügung, letztere ermöglichen die Integration optionaler, spezialisierter Funktionen und damit eine Anpassung der Lernplattform an die Erfordernisse einer bestimmten institutionellen Einsatzumgebung.

Vor dem Hintergrund der vorausgehend dargestellten Problemfelder verortet sich die Entwicklung der CampusSource Engine, die an verschiedenster Stelle auf Open Source Software zurückgreift und ihrerseits unter einer Open-Source-Lizenz zur Verfügung gestellt wird. Für Open Source Software sind dabei vor allem die folgenden drei Merkmale[1] charakteristisch: erstens liegen ihre Quelltexte öffentlich in einer für den Menschen lesbaren und verständlichen Form vor, zweitens darf sie beliebig kopiert, genutzt, und verbreitet werden, drittens darf sie verändert und in der veränderten Form weitergegeben werden. Die freie Zugänglichkeit des Quellcodes einer Software ermöglicht neben ihrer Verifikation insbesondere eine Weiterentwicklung durch Dritte, die in den meisten Fällen faktisch nicht existenten Lizenzkosten tragen zu einer weiten Verbreitung der Software bei. Hieraus resultieren diverse wünschenswerte Effekte: „Mit der weiten Verbreitung werden Fehler schneller gefunden und bei entsprechender Fachkompetenz können sie von den AnwenderInnen behoben werden, da die Voraussetzungen gegeben sind, den Quellcode ändern zu können und zu dürfen. Dies führt zu einer schnellen Qualitätssteigerung im Vergleich zu proprietärer Software bei der AnwenderInnen, die einen Fehler berichten, letztendlich nur auf ein neues Update hoffen können." [CampusSource].

Dem Einsatz geeigneter Open-Source-Produkte im Kontext der Entwicklung einer Lernplattform stehen gemäß den vorausgehenden Betrachtungen keine prinzipiellen Bedenken entgegen, vielmehr erscheint er sogar wünschenswert. Auch die Veröffentlichung der Software unter einer Open-Source-Lizenz ist in Zeiten einer angespannten Haushaltslage der Hochschulen sinnvoll und vermag die Nachhaltigkeit der Entwicklungsarbeit sicherzustellen: „Indem anderen Bildungseinrichtungen die Software zur Verfügung gestellt wird, können Dritte auf vorhandener Software aufbauen und neue Mittel fließen in die Weiterentwicklung existierender Systeme statt in eine Parallelentwicklung. Das Überleben der Software ist so auch in finanziell angespannten Zeiten gesichert. Weiterhin können die finanziellen Ressourcen des Landes und des Bundes sinnvoll und zielgerichtet eingesetzt werden. Nur indem die einzelnen Kräfte gebündelt werden, kann ausreichend Stärke erzielt werden, um im weltweiten Wettbewerb zu bestehen." [CampusSource].

Komponenten und Dienste als Bausteine einer Lernplattform

Der modulare Aufbau eines Systems wird im Kontext objektorientierter Softwareentwicklung durch Architekturschemata ermöglicht, deren Bausteine von einander unabhängige und wiederverwendbare Komponenten sind, die Organisation der Plattform in einzelne Komponenten stellt eine Grundlage für deren Erweiterbarkeit und Anpassbarkeit dar. Komponenten zeichnen sich durch die folgenden charakteristischen Eigenschaften aus:

- In einem objektorientierten Softwareentwurf beschreiben fachliche Komponenten die einzelnen funktionalen Module einer Lernumgebung (z.B. Authentifizierung / Autorisierung, Anbindung von Content Repositories, …). Solche Geschäftskomponenten können sich ihrerseits auf technischer Ebene aus mehreren Software-Komponenten zusammensetzen, die auf Ebene der Module einer Lernplattform somit nicht unbedingt über eine unmittelbare Entsprechung[2] verfügen.

- Komponenten gliedern sich in eine Rahmenarchitektur ein und verfügen über wohldefinierte öffentliche Schnittstellen, sie sind zueinander interoperabel. Die öffentliche Schnittstelle einer Komponente stellt den alleinigen Zugang zu den von ihr angebotenen Funktionalitäten dar.

- Eine Komponente kann verschiedene Sichten bzw. Zugriffsmöglichkeiten auf die von ihr zur Verfügung gestellten Funktionalitäten anbieten, hierunter fallen insbesondere – aber nicht ausschließlich – verschiedene Formen von Benutzerschnittstellen (z.B. Web-Oberflächen, Zugang über mobilfunkbasierte Dienste wie SMS, WAP).

[1] Vgl. http://de.wikipedia.org/wiki/Open_source

[2] Unter einem Modul kann gemäß diesen Feststellungen eine Komponente hohen Abstraktionsgrades bzw. ein Subsystem verstanden werden. Gebräuchlich ist auch der Begriff des Geschäftsobjekts, vgl. [Backschat & Gardon (2002), S. 6].

Eine komponentenorientierte Rahmenarchitektur eröffnet die Möglichkeit zur Entwicklung einer dienstbasierten, verteilten Lernplattform. Dienste sind – in Abgrenzung zu „klassischen“ Komponenten – eigenständig operierende Applikationen, die von einem Programm zur Laufzeit in einem Netzwerk lokalisiert und über öffentlich bekannte Schnittstellen verwendet werden können (vgl. Gehrke et al. (2002)). Die einzelnen Bestandteile einer Lernplattform lassen sich in Form von Diensten also auf verschiedene Server verteilen. Auf diese Weise kann eine erhöhte Ausfallsicherheit des Gesamtsystems erzielt werden, des Weiteren ist die Lastverteilung auf gleichartige Dienste mit geringem Aufwand möglich. Darüber hinaus können einzelne Institutionen bzw. Akteure spezialisierte Dienste zur Integration in verschiedenen Lernumgebungen anbieten. Beispielsweise könnte ein von der Universität Dortmund bereitgestellter Dienst zur Literaturrecherche auch von anderen Universitäten nahtlos in die jeweilige lokale Lernumgebung integriert werden.

Das zum gegenwärtigen Zeitpunkt populärste und am weitesten fortgeschrittene Konzept zur Realisierung dienstbasierter Systeme stellen so genannte Web Services dar. Bei einem *Web Service* handelt es sich um eine über standardisierte Protokolle (z.B. HTTP, SMTP) in einem Netzwerk erreichbare Applikation, deren Schnittstellen unabhängig von der zugrunde liegenden Implementierung unter Verwendung einer XML-basierten, standardisierten Sprache beschrieben werden können und die zur Interaktion mit anderen Applikationen ebenfalls auf ein sprach- sowie plattformunabhängiges, standardisiertes XML-Protokoll zurückgreift.

Nach Chappell & Jewell (2002) sind die folgenden Eigenschaften für Web Services charakteristisch (vgl. [Chappell & Jewell (2002), pp. 6-7]):

- Aufbau auf XML
  Sämtliche für Web Services relevanten Daten werden in Form wohldefinierter XML-Fragmente repräsentiert. Insbesondere kommt kein proprietäres (Binär-)Format zum Einsatz, was die den Web Services zu eigene Plattformunabhängigkeit ggf. einschränken würde.
- Geringe Kopplung
  Zwischen Client und Web Service besteht eine geringe Kopplung, was eine weitestgehend unabhängige Fortentwicklung beider Komponenten ermöglicht sowie die Möglichkeiten zu deren Wiederverwendung und Wartbarkeit verbessert.
- Geringe Granularität auf Ebene öffentlicher Schnittstellen
  Web Services stellen Klienten eine Schnittstelle zur Verfügung, die von spezifischen Details ihrer Implementierung abstrahiert. In gewissem Sinne realisieren sie eine Fassade für die zugrunde liegende, komplexe Geschäftslogik.
- Unterstützung synchroner oder asynchroner Operationsmodi
  Klienten können synchron oder asynchron mit einem Web Service kommunizieren. Nach Chappell & Jewell (2002) stellt die Möglichkeit zum asynchronen Datenaustausch einen wesentlichen Faktor dar, wenn eine geringe Kopplung der beteiligten Komponenten erzielt werden soll, vgl. [Chappell & Jewell (2002), p. 7].
- Unterstützung von Remote Procedure Calls (RPCs)
  Web Services stellen einen allgemeinen Mechanismus für RPCs zur Verfügung und vereinheitlichen so beispielsweise den Zugriff auf EJB- oder .NET-Komponenten.
- Möglichkeit zum Austausch von Dokumenten
  In XML codierte Dokumente können zwischen Web Services ausgetauscht werden.

Die von Web Services bereit gestellten Funktionalitäten zur Realisierung verteilter Softwaresysteme sind indes nicht neu: Technologien wie z.B. CORBA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model) und Jini verfolgen äquivalente Zielsetzungen, wurden nach Gehrke et al. (2002) aber *„nicht explizit für die Kommunikation über das Web entwickelt“* [Gehrke et al. (2002), S. 11] und sind einer Web Service Architektur in punkto Sprach- bzw. Plattformunabhängigkeit unterlegen.

Unter einer Lernumgebung kann nun gemäß den vorausgehenden Betrachtungen eine Bündelung von spezialisierten Diensten verstanden werden, die über wohldefinierte Schnittstellen miteinander interagieren können. Benutzer können auf vielfältige Weise mit der Lernplattform interagieren (s.u.) – auch wenn eine über einen Browser zugängliche webbasierte Benutzerschnittstelle, die in

Form eines eine personalisierte Sicht auf eine Auswahl der Dienste der Lernumgebung bereitstellenden Portals ausgeführt sein sollte, im Regelfall die am häufigsten genutzte Schnittstelle darstellt. Nicht jeder Dienst der Lernumgebung muss sich auf Ebene des Portals wieder finden, neben der durch ein Web-Portal beschriebenen Sicht sind auch andere Formen der Darstellung von Diensten möglich.

Standards für Entwurf und Implementierung

Die Rahmenarchitektur einer Lernplattform sollte sich zur Gewährleistung größtmöglicher Interoperabilität wo immer möglich an weithin akzeptierten Standards orientieren. Für eine Vielzahl von Komponenten existieren bereits standardisierte Schnittstellenspezifikationen, beispielsweise definiert JAAS (Java Authentication and Authorization Service) für Module der Zugriffskontrolle eine von konkreten Implementierungen (z.B. Kerberos, LDAP) abstrahierende Schnittstelle. Stellen die Komponenten einer Lernplattform die für ihr Aufgabengebiet existenten Standard-Schnittstellen zur Verfügung, so kann die Plattform leicht an die spezifischen Erfordernisse einer Einsatzumgebung angepasst werden. Die Verwendung standardisierter öffentlicher Schnittstellen auf Ebene der Komponenten wirkt sich außerdem günstig auf deren Substituierbarkeit aus.

Sowohl für die Architektur einzelner Komponenten als auch für deren Zusammenspiel in einer Lernplattform existieren Standards und gängige Verfahrensweisen unterschiedlicher Detailtiefe, die sowohl in Entwurfs- und Implementierungsphasen Berücksichtigung finden sollten:

- Eine Vielzahl von gut dokumentierten Entwurfsmustern ermöglicht es, erfolgreiche Designs und Architekturen objektorientierter Software wiederzuverwenden, eine dem zu lösenden Entwurfsproblem angemessene Anwendung solcher Muster kann auf entscheidende Art und Weise dazu beitragen, die Wartbarkeit, Wiederverwendbarkeit und Erweiterungsfähigkeit eines Software-Systems sicherzustellen. Auch wenn sich die teilweise eng an Sprachmittel[3] moderner objektorientierter Hochsprachen gekoppelte Anwendung von Entwurfsmustern mittlerweile in vielen Softwareprojekten etabliert hat und in diesem Sinne also „Gemeingut“ darstellen mag , soll hier dennoch auf die für die Softwaretechnik richtungweisende Arbeit der *Gang of Four* verwiesen werden: die von Gamma et al. (1996) beschriebenen Entwurfsprinzipien stellen nach wie vor eine solide Basis für das Design objektorientierter Software dar[4].

- Schichtenmodelle der Architektur (z.B. MVC – Model, View, Controller) gewährleisten eine wünschenswerte funktionale Trennung auch innerhalb einzelner Komponenten. Schichtenmodelle können im Sinne komplexer, nicht elementarer Entwurfsmuster höheren Abstraktionsniveaus verstanden werden und bilden einen de facto Standard[5] für den Entwurf komplexer Softwaresysteme.

- Durch J2EE / Enterprise JavaBeans (EJB) wird ein umfassender Standard für komponentenbasierte Client-/Server-Applikationen definiert, der sowohl Schnittstellenspezifikationen als auch Aspekte der Systemarchitektur adressiert. Eine J2EE-konforme Realisierung der Plattform trägt zur Sicherung der Erweiterbarkeit, der Integrationsfähigkeit in verschiedene Einsatzumgebungen und der Skalierbarkeit bei. Verfahrensweisen des Bereitstellens von Komponenten als eigenständige Dienste werden im Kontext der bereits erwähnten Web Services beschrieben, der ihrerseits eine breite Unterstützung durch J2EE erfahren.

[3] In Java spiegelt beispielsweise das Konstrukt der Interface-Klassen die puristische Anwendung des Entwurfsmusters *Facade* wider.
[4] Gleichwohl sei an dieser Stelle auch auf die mit dem Begriff der *Pattern Trap* bezeichnete Gefahr einer zu sehr an Entwurfsmustern (und eben nicht an der vorliegenden Problemstellung) orientierten Verfahrensweise hingewiesen: Muster sollten stets dann zum Einsatz kommen, wenn sie auf ein vorliegendes Problem passen – in keiner Weise sollte ihre Anwendung zum Selbstzweck geraten.
[5] vgl. z.B. http://de.wikipedia.org/wiki/MVC

- Den Autoren (Lehrenden und Lernenden) soll ein einfacher und komfortabler Mechanismus zur Verfügung gestellt werden, um den von ihnen entwickelten Content in die Plattform zu transferieren. Hierzu soll kein proprietäres Programm eingesetzt werden sondern ein interoperabler Standard, der von allen gängigen Betriebssystemen unterstützt wird. Dieser Standard heißt WebDAV (siehe u.a. [Dusseault (2004)]). Die Bezeichnung WebDAV steht für "Web-based Distributed Authoring and Versioning. Der WebDAV Standard ist in [rfc2518] beschrieben und wurde ergänzt um den Bereich Versioning in [rfc3253]. An der Entwicklung waren u.a. die Firmen Microsoft, Netscape und Novell beteiligt. Die serverseitige Verwendung von WebDAV ermöglicht u.a. quasi ein einfaches clientseitiges Mounten von Verzeichnissen auf dem Server, so dass ein Client die Verzeichnisse wie seine lokalen Verzeichnisse benutzen kann. Neben den gewohnten Datei- und Verzeichnisoperationen stellt WebDAV die Funktionalitäten Locking und Versioning zur Verfügung. Programme wie zum Beispiel Microsoft Office, das von manchen Autoren benutzt wird, nutzen das Locking automatisch. WebDAV benutzt HTTP als Transportprotokoll und wird somit durch viele Firewalls nicht gestört.

- Gegenwärtig erfahren Portalsysteme im Rahmen der „Portlet Specification“ (JSR 168) eine Standardisierung im Java Community Process. Die Berücksichtigung dieses Standards kann der zu realisierenden Lernplattform eine Vielzahl zusätzlicher Module erschließen, gleichermaßen könnten die Module der Plattform von jedem JSR168-fähigen Portalserver verwendet werden. Sofern sich der Entwurf der Plattform auf ein J2EE-konformes Schichtenmodell stützt, können die einzelnen Komponenten leicht als Portlet gemäß JSR168 verfügbar gemacht werden.

Über die Orientierung an den oben angesprochenen Standards hinaus können Entwurf und Implementierung einer Lernplattform von einer Vielzahl frei verfügbarer Produkte und Werkzeuge profitieren, die teilweise sehr mächtige Funktionalitäten für die Entwicklung webbasierter Systeme bereitstellen und häufig Referenzimplementierungen bestimmter Standards darstellen oder aber selbst de facto Standards definieren.

Grundzüge der CampusSource Engine-Architektur

Eine komponentenorientiert entwickelte Lernplattform lässt sich in diverse logische Schichten unterteilen, die jeweils einen spezifischen Aufgabenbereich innerhalb der Applikation abdecken und dabei stets nur mit ihren unmittelbar benachbarten Schichten über wohldefinierte Schnittstellen kommunizieren, vgl. [Backschat & Gardon (2002), S. 7, 9]. Diverse Architekturschemata beschreiben die Aufteilung eines Softwaresystems in solche Ebenen auf unterschiedlichem Abstraktionsniveau: als prominenteste Beispiele seien Client/Server-Systeme und *Three Tier* Anwendungsarchitekturen genannt – letztere begegnen den für „klassische“ zweischichtige Client/Server-Systeme typischen Problemen reduzierter Skalierbarkeit[6] und aufwändiger Wartbarkeit[7].

Die Architektur der CampusSource Engine folgt – global betrachtet – einem typischen Dreiebenen-Modell, das sich aus einer die Benutzerschnittstelle vorhaltenden Client-Schicht, einer die Geschäftslogik verkapselnden Mittelschicht (*Business Tier)* und schließlich einer die Persistenz der Anwendungsdaten gewährleistenden Datenschicht (*Back-End-Tier)* zusammensetzt. Die konkrete Ausformulierung dieser Architektur wird dabei maßgeblich von der Enterprise JavaBean Spezifikation beeinflusst, die einen Industriestandard für die Entwicklung komponentenbasierter Software im Umfeld von Applikationsservern und anderen Middleware-Produkten darstellt.

[6] Backschat & Gardon (2002) identifizieren im Kontext verteilter Datenbankanwendungen eine „*hohe Netzbelastung* infolge einer großen Zahl von erforderlichen Kommunikationsschritten zwischen dem Client, der SQL-Anweisungen absendet, und dem Datenbank-Server, der in der Regel größere Ergebnismengen absendet und dabei viel mehr Daten vom DB-System an den Client transferiert als der Benutzer benötigt.“ [Backschat & Gardon (2002), S. 8].

[7] Die Schwierigkeiten werden offensichtlich, wenn man beispielsweise an die Aktualisierung einer großen Menge proprietärer Clients zur Gewährleistung eines einheitlichen Versionsstandes denkt.

Bestandteile der EJB-Architektur

Auf oberster Ebene unterteilt sich das EJB-Architekturmodell in die Bereiche des EJB-Servers, des EJB-Containers, der Enterprise Beans und der Clients. Diese Elemente können hinsichtlich ihrer wesentlichen Eigenschaften wie folgt beschrieben werden, vgl. [Backschat & Gardon (2002), S. 24-25]:

- Der **EJB-Server** verwaltet den EJB-Container, die Gewährleistung von Verfügbarkeit und Skalierbarkeit, Verbindungs-Management, Lastausgleich und Fehler-Management sowie die Anbindung von (lokalen) IuK-Subsystemen gehören zu seinem Aufgabenbereich.

- Die Laufzeitumgebung für installierte Komponenten (i.d.R. Enterprise Beans) wird vom **EJB-Container** zur Verfügung gestellt, der neben dem Management des Lebenszyklus einzelner Bean-Exemplare diesen insbesondere standardisierte Zugriffsmöglichkeiten auf so genannte Enterprise Services zur Verfügung stellt (hierunter fallen z.B. Datenbankzugriffe sowie Messaging-, Namens- und Transaktionsdienste). Der EJB-Container ermöglicht als grundlegende Funktionalität weiterhin den Zugriff (entfernter) Clients auf die von ihm vorgehaltenen Komponenten.

- **Enterprise Beans** sind Server-Komponenten, welche der Verkapselung der Applikationslogik dienen. Das EJB-Architekturmodell kennt drei unterschiedliche Typen von Enterprise Beans: persistente Geschäftsdaten werden von Entitiy Beans abgebildet, die einen transaktionsgesicherten Zugriff mehrerer Clients auf ein und dasselbe Entity-Objekt ermöglichen. Die Speicherung eines Entity Beans (z.B. in einer relationalen Datenbank) kann für den Applikationsentwickler transparent vom EJB-Container übernommen werden (Container-Managed Persistence, CMP) oder der Entity Bean selbst obliegen (Bean-Managed Persistence, BMP), vgl. [Backschat & Gardon (2002), S. 34].
  Session Beans dienen der Modellierung von Geschäftsprozessen, sie werden i.d.R. im Zusammenhang mit einer entsprechenden Anfrage eines Clients erzeugt und (instanziiert) und stellen über (öffentliche) Methoden Zugriffsmöglichkeiten auf die (serverseitige) Geschäftslogik einer Anwendung zur Verfügung. Man unterscheidet Stateless und Stateful Session Beans voneinander: Stateless Session Beans beinhalten keine Client-spezifischen Informationen, welche über die bei einem Methodenaufruf übergebenen Parameter hinausgingen – insbesondere halten sie keine Zustandsinformationen über mehrere Methodenaufrufe hinweg vor. In Abgrenzung dazu können Stateful Session Beans clientspezifische Zustandsinformationen über mehrere Methodenaufrufe hinweg vorhalten, diese bleiben jedoch nicht über die Lebensdauer des Session Beans hinaus persistent.
  Die dritte Ausprägung von Enterprise Beans wird letztlich von den so genannten Message-Driven Beans gebildet, die über einen Messaging-Dienst empfangene Nachrichten asynchron verarbeiten können. In Abgrenzung zu Entity Beans und Session Beans erfolgt die Interaktion mit Message-Driven Beans indirekt, insbesondere führt ein Client keine unmittelbaren Methodenaufrufe aus.
  Für den Zugriff eines Clients auf Enterprise Beans stehen unterschiedliche Schnittstellen zur Verfügung: außerhalb des EJB-Containers residierende Clients greifen über das Remote Home-Interface auf Methoden zum Management des Lebenszyklus einer Entity Bean zu, das Remote Komponenten-Interface stellt die Geschäftsmethoden einer Entity Bean zur Verfügung. Für lokale Clients (also solche, die innerhalb desselben EJB-Containers residieren wie die betroffene Enterprise Bean selbst) erfüllen das Lokale Home-Interface und das Lokale Komponenten-Interface äquivalente Aufgaben, vgl. [Backschat & Gardon (2002), S. 39-49].

- Der Zugriff auf Enterprise Beans kann von **Clients** verschiedener Typen aus erfolgen, denkbar sind z.B. browser-basierte *Thin Clients*, eigene Applikationslogik beinhaltende *Fat Clients*, Servlets oder andere Enterprise Beans. Es sei an dieser Stelle erwähnt, dass der Zugriff auf Enterprise Beans auch im Kontext einer Rahmenarchitektur für verteilte Dienste (z.B. CORBA, WebServices) erfolgen kann (s.u.).

Auf welche Art und Weise die Nutzung der von Enterprise Beans angebotenen Geschäftslogik unter Vermeidung einer engen strukturellen Kopplung durch lokale und entfernte Klienten erfolgen kann, ist u.a. Thema der folgenden Unterkapitel.

Entkopplung der Präsentationsschicht mittels XMLC

Die Präsentationsschicht komplexer Web-Anwendungen beinhaltet häufig Inhalte, die dynamisch unter Verwendung der serverseitigen Geschäftslogik erstellt werden. Häufig kommen zum Erzeugen solcher Ausgaben Java Server Pages (JSP) zum Einsatz, die eine Pull-Strategie der Seitengenerierung verfolgen: Java-Code und spezielle JSP-Aktionen wie z.B. der Aufruf einer Funktion aus einer Tag Library werden in eine Markup-Seite eingebettet und greifen auf die Geschäftslogik der Web-Anwendung zu. Nachteilig an dieser Verfahrensweise ist, dass sie bei unstrukturierter Anwendung leicht Präsentations- und Geschäftslogik miteinander koppelt und damit nicht zuletzt die Wiederverwendbarkeit und Wartbarkeit einzelner Bestandteile der Applikation drastisch einschränkt.

Der im Kontext des Enhydra-Projektes entwickelte XML Compiler (XMLC) verfolgt vor dem Hintergrund der potenziellen Nachteile von JSP einen anderen Ansatz: XML-basierte Markup-Seiten werden mit Hilfe des XMLC zu Java-Klassen compiliert , auf die dann mit Hilfe des Document Object Model (DOM) zugegriffen werden kann. Die Generierung dynamischer Seiten folgt somit einer Push-Strategie, die in drei Schritte unterteilt werden kann:

1. In einer Entwurfsphase werden Bereiche einer Markup-Seite identifiziert, die dynamische Elemente beinhalten. Diesen Bereichen werden eindeutige Namen zugeordnet, welche innerhalb der Markup-Seite unter Verwendung entsprechender ID-Attribute bekannt gemacht werden.

2. Die um ID-Attribute angereicherte Markup-Seite wird mittels des XMLC kompiliert. Es entsteht eine Java-Klasse, die eine DOM-Repräsentation der Markup-Seite sowie getter- und setter-Methoden für jeden mit einem eindeutigen Namen versehenen Bereich beinhaltet.

3. Zur Laufzeit der Applikation kann die im vorausgehenden Schritt erzeugte Java-Klasse mittels der zu diesem Zweck generierten Methoden um dynamische Inhalte erweitert werden. Die DOM-Repräsentation der Klasse lässt sich dabei als Markup-Seite jederzeit an den Klienten zurücksenden.

Die Vorteile der oben skizzierten Verfahrensweise sind nach Young (2002) vielgestaltig, herausgehoben seien die folgenden Aspekte:

- In eine Markup-Seite wird niemals Java-Code eingebettet, es wird ausschließlich von Tags gebrauch gemacht, die der Syntax der Markup-Sprache entsprechen. Auf diese Weise werden eine Entkopplung der Geschäftslogik von der Präsentationsschicht ermöglicht sowie wohlgeformte Markup-Dokumente sichergestellt.

- Die Trennung von Präsentationsschicht und Geschäftslogik trägt positiv zur Wartbarkeit und Erweiterbarkeit der Web-Anwendung bei.

- Designer und Programmierer können weitestgehend unabhängig von einander an einer Web-Applikation arbeiten, die Schnittmenge ihrer Aufgabengebiete beschränkt sich auf die Definition geeigneter ID-Attribute für dynamisch zu erzeugende Inhalte.

Abschließend für diesen Abschnitt sei noch einmal explizit darauf hingewiesen, dass XMLC mit nahezu jeder XML-basierten Sprache zusammenarbeiten kann und somit eine viel versprechende Technologie zur Erzeugung multipler Sichten auf die Geschäftslogik einer Anwendung darstellt.

Personalisiertes Angebot von Diensten in einem Portalsystem

Portalsysteme ermöglichen es, verschiedene voneinander unabhängige Dienste zu einer Gesamtanwendung zusammenzufassen und diese über eine server-basierte Nutzerschnittstelle auf einheitliche Art und Weise zugänglich zu machen, vgl. [it-fws (2003)]. Die Implementierung und

Umsetzung solcher Portalsysteme konnte sich bis dato an keinem weithin akzeptierten Standard orientieren und war somit nahezu notwendigerweise abhängig von proprietären Technologien einzelner Portalserver. Die im Herbst 2003 unter der Bezeichnung JSR 168 im Java Community Process (JCP) entstandene Spezifikation schafft hier Abhilfe und definiert zum ersten Mal einen weithin akzeptierten Standard für Portale.

Abbildung 2 stellt den Aufbau eines Portalsystems vereinfacht dar: auf Ebene des Klienten finden sich einzelne Dienste (wie z.B. ein Literatur-Recherche-Dienst) als Bausteine innerhalb einer einheitlichen Nutzerschnittstelle wieder. Die von diesen so genannten Portlets angebotenen Funktionalitäten werden von eigenständigen Komponenten zur Verfügung gestellt, die in die Laufzeitumgebung eines Portlet Containers eingebettet sind. Der Portlet Container steuert den Lebenszyklus einzelner Portlets und stellt einen Kontext zur Kommunikation zwischen einzelnen Portlets bereit, er hat nicht die Aufgabe, die Darstellung der einzelnen Portlets zu einer Gesamtansicht zu aggregieren – diese Aufgabe wird vom Portlet Server übernommen.

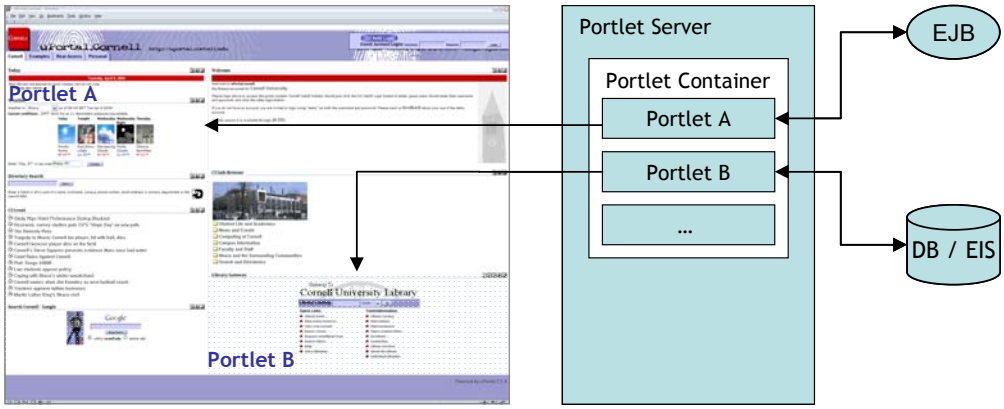

Abbildung 2: Aufbau eines Portalsystems

Die JSR-168 Spezifikation ist eng mit J2EE verwoben, der Portlet-Standard und die zugehörige API orientieren sich stark an der dort definierten Servlet API: typische Konzepte wie z.B. das Request/Response-Prinzip, die Sessionverwaltung und die Nutzung eines gemeinsamen Kontextes finden sich hier wieder, vgl. [it-fws (2003)]. JSR 168 kann als eine Schicht angesehen werden, die auf der Servlet-Spezifikation 2.3 aufsetzt, die Integration der beiden Technologien vollzieht sich nahtlos.

Weiterhin von Interesse ist die durch den Standard *Web Services for Remote Portlets* (WSRP) ermöglichte plattformunabhängige Integration entfernter Inhalte und Applikationen: dieser bildet eine Brücke zu den im folgenden Abschnitt näher betrachteten Web Services.

Integration externer Dienste als WebServices

WebServices wurden in diesem Dokument bereits als viel versprechende Rahmenarchitektur zur Realisierung dienstbasierter Systeme genannt und ihre wesentlichen Charakteristika kurz skizziert. Um zu verstehen, wie die CampusSource Engine von externen Diensten Gebrauch machen kann, ist eine kurze Darstellung der Technologien hilfreich, auf denen die WebService-Spezifikation basiert:

- Das **Simple Object Access Protocol (SOAP)** definiert, auf welche Art und Weise Web Services untereinander bzw. beliebige Applikationen mit Web Services Nachrichten austauschen: „SOAP ermöglicht sowohl synchrone als auch asynchrone Kommunikation, die von XML-basierten Remote Procedure Calls (RPCs) 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“ [Gehrke et al. (2002), S. 6-7].

- Die **Web Service Description Language (WSDL)** dient zur Beschreibung der Schnittstellen eines Web Service und ermöglicht es einem Klienten zu „verstehen“, auf

welche Weise ein spezifischer Web Service zu verwenden ist. Zu diesem Zweck lassen sich u.a. Eingabeparameter und Rückgabewerte eines Funktionsaufrufs sowie Protokollbindungen plattform- und sprachunabhängig beschreiben.

- Mit Hilfe von **Universal Description, Discovery and Integration (UDDI)** werden Informationen über Funktion und Verfügbarkeit eines Web Service beschrieben, so dass ein Klient unter Verwendung einer zentralen UDDI-Registry nach spezifischen, von ihm benötigten Diensten suchen kann und – sofern geeignete Dienste verfügbar sind – Verweise auf diese und die sie beschreibenden WSDL-Dokumente in Form von URLs erhält. Die Verwendung einer WSDL Registry ist nicht zwingend erforderlich: ein Klient kann sich auch direkt mit einem ihm bekannten Web Service verbinden, ohne diesen zuvor über UDDI anfragen zu müssen.

Abbildung 3 verdeutlicht das Zusammenspiel der soeben genannten Technologien anhand eines simplen Einsatzszenarios, sämtlichen Kommunikationsprozessen liegt dabei das XML-basierte SOAP zugrunde.

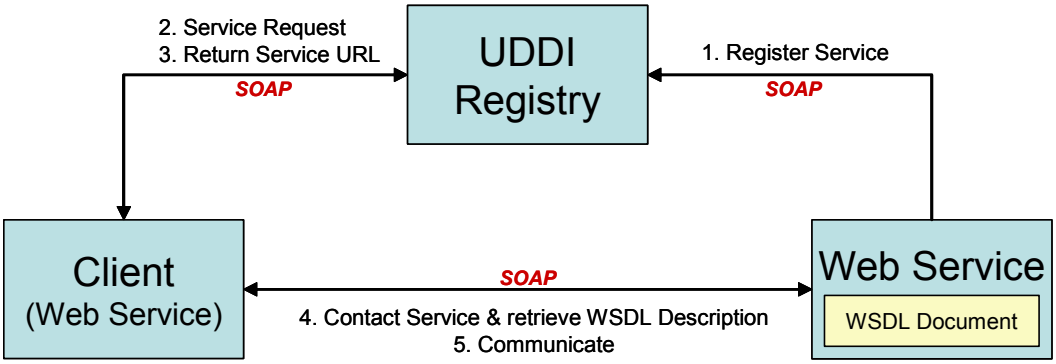

Abbildung 3: Zusammenspiel von SOAP, UDDI und WSDL

In Schritt 1 hinterlegt ein Web Service bei einer UDDI Registry zunächst Informationen, die von potenziellen Klienten zur Lokalisierung des Service verwendet werden können. Der Datenbestand der UDDI Registry kann von Klienten durchsucht werden, um geeignete Web Services ausfindig zu machen (Schritt 2): hat ein Klient Interesse an einem der UDDI Registry bekannten Service, so erhält er von dieser dessen URL (Schritt 3). Für die sich nun anschließenden Kommunikationsprozesse wird die UDDI Registry nicht mehr benötigt. Der Klient erfragt vom lokalisierten Web Service zunächst die zugehörige WSDL Description (Schritt 4) und hat damit alle zum Aufruf der angebotenen Funktionalitäten (Schritt 5) benötigten Informationen zur Verfügung.
Analog zur soeben geschilderten Verfahrensweise kann die CampusSource Engine unter Verwendung von WebServices beispielsweise externe Content Repositories oder Literatur-Recherchedienste nahtlos in eine standortspezifische Lernumgebung integrieren.
Die Verbindung zur vorausgehend diskutierten EJB-Architektur ist nahe liegend: WebServices können genutzt werden, um die Remote Interfaces einer Enterprise Bean auf standardisierte Art und Weise für eine Vielzahl von Klienten nutzbar zu machen.

Die CampusSource Engine als Anwendung innerhalb eines Applikationsservers

Im vorausgehenden Teil dieses Beitrags wurden wesentliche Basistechnologien der CampusSource Engine vorgestellt. Aufgabe dieses Unterkapitels ist es, das Zusammenspiel dieser Elemente zusammenfassend darzustellen. Abbildung 4 zeigt die CampusSource Engine als Applikation in einem EJB-Server (hier exemplarisch: JBoss), die oben genannten Technologien finden sich

vornehmlich als in einer Drei-Schichten-Architektur verorteten Komponente innerhalb des EJB-Containers[8] wieder:

- Innerhalb der Präsentationsschicht (Presentation Tier) nutzt ein für die Bereitstellung der Nutzerschnittstelle verantwortliches „Mediator"-Servlet XMLC und Portlets zur Erfüllung seiner Aufgabe. Es sei erwähnt, dass der Entwurf der CampusSource Engine an dieser Stelle Möglichkeiten[9] zum Erstellen alternativer Sichten berücksichtigt und somit insbesondere keine Festlegung auf einen XMLC-basierten Ansatz erfolgt.

- Session EJBs sind die zentralen Elemente der die Geschäftslogik der CampusSource Engine verkapselnden mittleren Schicht (Business Tier). Sie sind über ihr Home Interface für die Präsentationsschicht zugänglich und erlauben über das Remote Interface außerhalb des EJB Containers (bzw. EJB Servers) liegenden Applikationen den Zugriff auf die von ihnen vorgehaltenen (öffentlichen) Funktionalitäten. Message-Driven EJBs sind ebenfalls Teil der Mittelschicht, innerhalb der CampusSource Engine liegt eine mögliche Anwendung in der Bereitstellung eines asynchron operierenden Logging-Mechanismus.

- Die Persistenzschicht (Persistence Tier) setzt sich aus Entity EJBs zusammen, deren Aufgabenspektrum sich von der Speicherung plattformspezifischer (Konfigurations-)Daten bis hin zur Anbindung externer Dokumentenserver erstreckt. Sowohl Strategien der Bean-Managed Persistence als auch der Container-Managed Persistence können sinnvoll zum Einsatz kommen, die Verwendung spezialisierter objektrelationaler Mapping-Werkzeuge (ORM) wie z.B. Hibernate[10] bietet sich in diesem Kontext an.

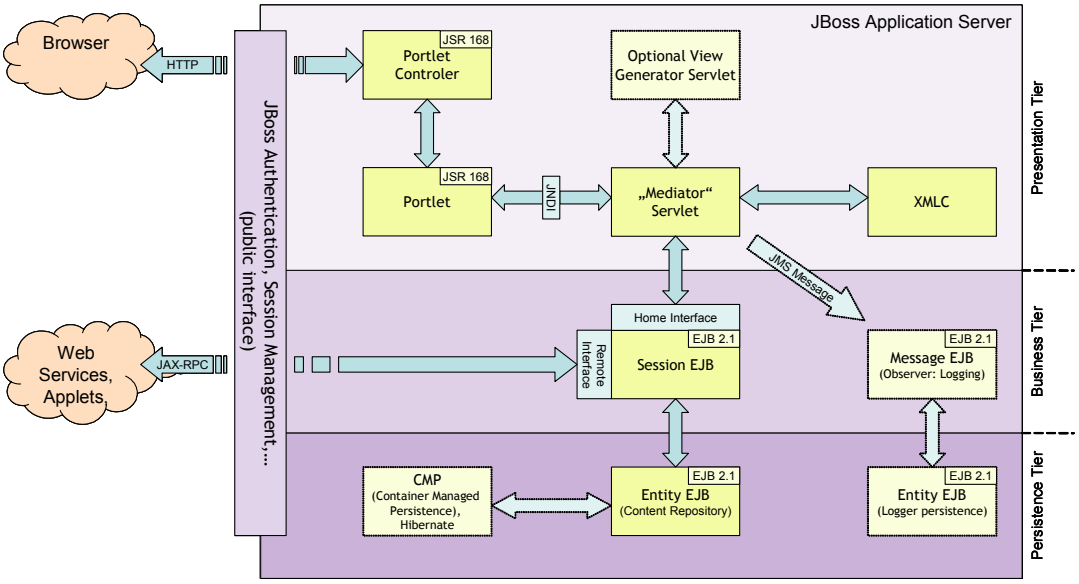

Abbildung 4: Bestandteile der CampusSource Engine in einem Applikationsserver

Web Services nehmen im obigen Architekturschema eine Sonderrolle ein: sie werden als Schnittstellentechnologie angesehen, die gemäß den vorausgehenden Ausführungen einen standardisierten, implementierungsunabhängigen Zugriff auf die von der CampusSource Engine angebotenen Dienste ermöglichen.

[8] Beachte, dass der EJB Container in Abbildung 3 nicht explizit als eigenständiges Element innerhalb des EJB Servers dargestellt wird.

[9] Von Bedeutung ist an dieser Stelle etwa das Entwurfsmuster *Builder*, welches die Konstruktion eines komplexen Objekts (beispielsweise der Darstellung eines Dokuments innerhalb einer personalisierten Web-Schnittstelle) von dessen Repräsentation (z.B. als Element eines Content Repositiories oder – allgemeiner – als Rückgabeobjekt des Aufrufs einer Methode der Geschäftslogik) entkoppelt, vgl. [Gamma et al. (1996)].

[10] http://www.hibernate.org/

Zusammenfassung und Ausblick

Der vorliegende Beitrag hat sich bemüht, die Vorteile eines in weiten Teilen auf gut dokumentierten Standards beruhenden Entwurfs einer modularen Lernplattform herauszuarbeiten: zu diesem Zweck wurden grundlegende Technologien sowie einige Aspekte der Softwarearchitektur einführend vorgestellt und bezüglich ihrer Bedeutung für die CampusSource Engine und den aus ihrer Anwendung resultierenden Vorteilen betrachtet.

Literatur

[Alexander et al. (1997)] Alexander, Christopher; Ishikawa, Sara; Silverstein, Murray; Jacobson, Max; Fiksdahl-King, Ingrid; Angel, Shlomo (1977): A pattern language. New York: Oxford University Press.

[Backschat & Gardon (2002)] Backschat, Martin; Gardon, Otto (2002): Enterprise JavaBeans: Grundlagen – Konzepte – Praxis. Heidelberg, Berlin: Spektrum (2002).

[CampusSource] http://www.campussource.de/wir/

[Dusseault (2004)] Dusseault, Lisa (2004): *WebDAV - Next-Generation Collaborative Web Authoring*, Prentice Hall

[Chappell & Tyler (2002)] Chappell, David; Jewell, Tyler (2002): Java Web Services. O'Reilly.

[Gamma et al. (1996)] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1996): Entwurfsmuster: Elemente wieder verwendbarer objektorientierter Software. Bonn: Addison-Wesley.

[Gehrke et al. (2002)] Gehrke, Matthias; Meyer, Matthias; Schäfer, Wilhelm (2002): Eine Rahmenarchitektur für verteilte Lehr- und Lernsysteme. http://www.campussource.de/projekte/rahmenarchitektur.html

[rfc2518] HTTP Extensions for Distributed Authoring -- WEBDAV, http://www.ietf.org/rfc/rfc2518.txt

[rfc3253] Versioning Extensions to WebDAV, http://www.ietf.org/rfc/rfc3253.txt

[Young (2002)] Young, David H. (2002): A Friendly Game of Tug of War: XMLC vs. JSP. The Server Side, http://www.theserverside.com/articles/article.tss?l=XMLCvsJSP