Die Datenintegration zwischen HIS-LSF und universitären Lern- und Arbeitsplattformen – Ein Praxisbericht aus dem Locomotion Projekt*

DSOR Arbeitspapier WP0701

Hans Gossen[1] , Alexander Roth[1]

[1] Decision Support & OR-Lab, Universität Paderborn

{hago, roth}@uni-paderborn.de

Abstract: In den neueren Versionen des HIS-GX Moduls LSF gibt es einen Prototyp für SOAP-basierte Webdienste, die Suchanfragen im LSF-Datenbestand erlauben. Im Rahmen des Förderprojekts Locomotion an der Universität Paderborn haben wir diese Schnittstelle getestet. Hier präsentieren wir die Ergebnisse und unsere Erfahrungen.

1. Einleitung

Das HIS-GX Modul Lehre, Studium, Forschung (kurz: HIS LSF) dient als als Studieninformations- und Planungssystem und verwaltet u.a. auch Veranstaltungsinformationen und Teilnahmedaten, die in den in der Lehre eingesetzten IuK-Systeme teilweise sogar mehrfach erfasst werden müssen. Um in einer heterogenen IT-Infrastruktur zusätzliche Aufwände für Administration und Datenerfassung in der Durchführung der Lehre einsparen zu können, bedarf es einer Integration, ohne dabei die vorhandenen Vielfalt an didaktischen Arrangements einzuschränken (vgl. hierzu auch [KNW03] und [RSS04]). So könnten zentrale Informationen in Systemverbünden gemeinsam genutzt und neben Kostenersparnissen sogar weitere Synergieeffekte erzielt werden (vgl. [RH06]).

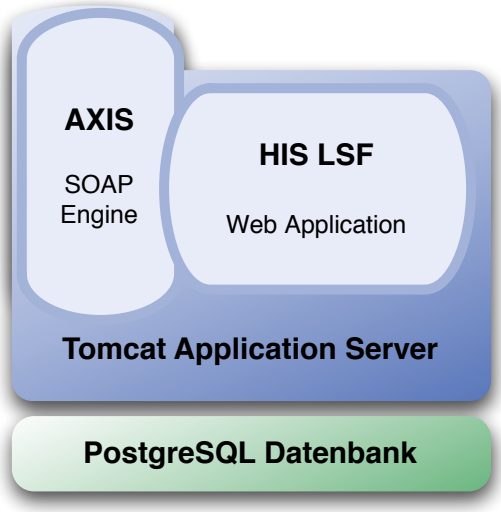
Eine Schnittstelle zwischen der HIS Verwaltungssoftware und den in der Lehre eingesetzten Werkzeugen muss daher unabhängig eines bestimmten Werkzeugs oder Programmiersprache sein, um nicht für jede weitere Plattform eine weitere Schnittstelle zu pflegen. Die SOAP-basierten Webdienste des HIS LSF Moduls erfüllt diese Anforderung. Die Schnittstelle ist sehr offen gehalten und entsprechend flexibel bzgl. der Daten und Strukturen, die hierüber abgefragt werden können. Umso schwieriger ist es, generelle Aussagen über die Einsetzbarkeit an allen Universitäten zu treffen. Individuell müssen insbesondere die jeweilige Plattform und das Szenario daraufhin untersucht werden, ob eine Datenintegration (das aktive *Daten abfragen*) das spezielle Problem löst oder ob eine darüber hinausgehende Funktionsintegration (z. B. das passive *aufgerufen werden*) notwendig ist (vgl. [RSS04]).

Im Rahmen des Paderborner Locomotion-Projektes haben wir ein Testinstallation des HIS LSF Moduls in der Version 9 betrieben, und die in diesem Papier vorgestellten Szenarien in Java und PHP5 prototypisch umgesetzt. Als Dokumentation stand uns ausschließlich das Media-Wiki der HIS zur Verfügung. Auf den folgenden Seiten werden wir nun versuchen, unsere auf dieser Basis gewonnenen Erkenntnisse im allgemeinen Umgang mit dieser Schnittstelle weiterzugeben, sowie auf dieser Basis Abschätzungen für eine allgemeine Alltagstauglichkeit zu treffen. Wir werden in Abschnitt 2 die grundlegende Architektur und Funktionsweise der Schnittstelle aufzeigen und eine Auswahl der für die Anbindung von Lernwerkzeugen wichtigen Dienste treffen. In Abschnitt 3 gehen wir kurz auf Aspekte der Client-Programmierung ein und skizzieren ein Grundgerüst in der Skriptsprache PHP5, mit dem die in Abschnitt 4 vorgestellten Beispielszenarien und Konfigurationen ausprobiert werden können. Abschließend geben wir in Abschnitt 5 eine zusammenfassende Beurteilung.

2. Grundlegende Architektur und Funktionsweise

Der Apache Tomcat Application-Server bildet die Laufzeitumgebung für das HIS LSF Modul, dessen Code als Servlet ausgeführt wird. Für die Verwaltung der Web-Services wird die Java-Implementierung der SOAP-Engine Apache

Low-Cost Multimedia Organisation and Production, ein prozessorientiertes Vorhaben der Universität Paderborn zur alltagstauglichen Erschließung und nachhaltigen Verankerung digitaler Medien in allen Bereichen ihrer Lehr- und Lernpraxis, siehe http://locomotion.uni-paderborn.de



HIS LSF läuft als Applikation in der Web-Container-Umgebung Apache Tomcat. Dabei greift es für die Abwicklung von Web-Services auf die Java-Implementierung der SOAP-Engine Apache AXIS zu.
Axis läuft dabei selbst als Java-Servlet innerhalb von Tomcat. Die dort registrierten Web-Services sind unter %SERVER_NAME/quisserver/services verfügbar.

Abbildung 1. Die Architektur von HIS-LSF

AXIS zurückgegriffen, die ebenfalls als Servlet im Tomcat läuft. AXIS bietet an, dass Java Applikationen direkt als JWS (Java Webservice) bereitgestellt werden können. Daraus generiert AXIS automatisch einen verwendbaren Webservice inklusive WSDL Beschreibung, die über das Netz ausgelesen werden kann, um in der aufrufenden Anwendung zur Laufzeit einen entsprechenden Client dynamisch zu erzeugen (vgl. bspw. Listing 1, Zeile 3-4).

Ein zentrales Element von LSF aus Sicht der SOAP-Schnittstelle ist das *LSFPublishModul*, welches flexible Anfragen an unterschiedliche Datenbanken und die Weiterverarbeitung der Ergebnisse in Form eines XML-Baums ermöglicht (vgl. Abb. 2). Auf dieser Basis können über das Modul nicht nur Berichte in Formaten wie PDF, RTF, HTML etc. generiert, sondern auch die Web-Services bedient werden.

In der Datei *modules.xml* müssen die states $axisbridge$, $soapsearch$ und $dbinterface$ aktiviert werden. Die Einträge sind in der Originaldatei bereits vorkonfiguriert, so dass nur noch das jeweilige Attribut $active =' y'$ gesetzt werden muss. Eine Auflistung aller registrierten Web-Services kann danach unter der Adresse %SERVER_NAME/quisserver/services abgerufen werden.

Die beiden für die Datenintegration wichtigen Dienste heißen *dbinterface* und *soapsearch*. Ihre Hauptmethoden mit ihren Parametern sind in Tabelle 1 aufgelistet. Ein erster Blick auf diese Parameter vermittelt den richtigen Eindruck, dass die Schnittstelle alles andere als selbsterklärend ist. Anstatt individuelle Web-Services für spezifische Abfragen anzubieten, können ausschließlich allgemeine Methoden genutzt werden, denen spezifische Abfragen als XML-Struktur übergeben werden müssen (*xmlParams*). Konkrete (aussagekräftige) Parameter werden nicht genutzt. Somit wird die Anwendungslogik der Methode nicht gekapselt, sondern dass Wissen um die internen Datenstrukturen und der Semantik muss in den aufrufenden Applikationen fest implementiert werden. Doch zunächst – und das ist die Vorbedingung – muss die Logik der Abfragen in den Konfigurationsdateien der Dienste definiert werden[1]. Hier finden sich schon einige Objekte, die über die Web Services abgerufen werden können[2]. Neue Objekte können dort hinzugefügt werden.

Die Definition der über SOAP abrufbaren Objekte werden in XML hinterlegt. Dazu wird a) eine SQL-Anweisung beschrieben, welche die für das Objekt benötigten Daten aus der Datenbank holt, und b) ein Mapping des SQL-Ergebnisses auf eine XML-Struktur, die als Ergebnis zurückgeliefert wird. Beispiele für diese Definitionen legen wir im Anhang bei.

[1]Die Konfigurationsdateien finden sich in /conf/dbinterface/soapgetdata.xml, ./soapsearch.xml und ./soapgetsearch.xml.

[2]Lt. HIS Wiki sind einige von ihnen *in Spezialmodulen für bestimmte Hochschulen konfiguriert und stehen somit nur dort zur Verfügung*. Unter "Allgemein verfügbare Objekte" ist nur der Typ *StudentMinimalType* aufgelistet.

Dienste	Methoden	Parameter	Beschreibung
soapsearch	getForm	String xmlParams	Auslesen des Aufbaus der LSF-Suchmasken
	search	String xmlParams	Suche nach Objekten über Felder, liefert Ausgangsbasis für die dbinterface Suche über IDs
dbinterface	getData	String className, String id	Zur Detaildarstellung von einzelnen Objekten, wenn die ID bekannt ist.
	getDataByParams	String className, String xmlParams	erweitert getData um konfigurierbare Abfrageparameter, flache XML-Struktur
	getDataXML	String xmlParams	erweitert getData um komplexere Abfrageparameter, tiefere XML-Struktur
	getDataSearch	String xmlParams	Anfragen über mehrere Objekte, z. B. Stichwortsuche über Veranstaltungen, Projekte und Einrichtungen

Tabelle 1. Überblick über wichtige HIS-LSF Web-Services

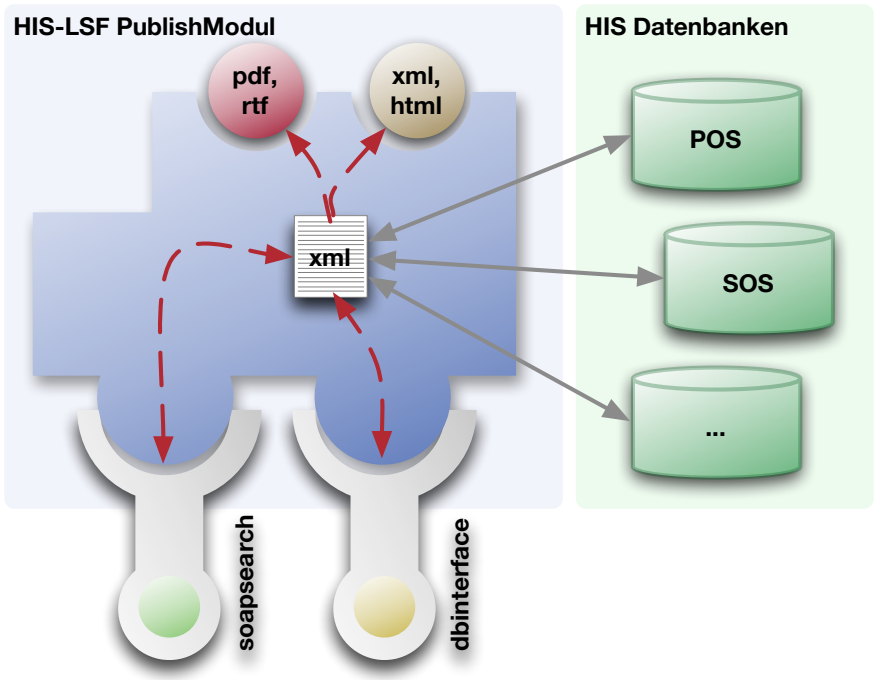


Abbildung 2. Das LSF-PublishModul ermöglicht flexible Anfragen an unterschiedliche Datenbanken und die Weiterverarbeitung der Ergebnisse in Form eines XML-Baumes. Die verfügbaren WebServices basieren auf dieser Funktionalität.

3. Hinweise zur Client-Programmierung

Die Programmierung von SOAP-Clients für die anzufragenden Anwendungen (Lern- und Arbeitswerkzeuge) ist relativ simpel und wird durch den durch die AXIS SOAP Engine generierten WSDL-Code unterstützt. Fast alle aktuellen Programmiersprachen bieten eine SOAP-Implementierung an, mit welcher ein Client dynamisch über die WSDL-Beschreibung zur Laufzeit erzeugt und genutzt werden kann. Ein Beispiel-Programm für Java ist im HIS Wiki unter dem Stichwort *LSF-Inhalte per SOAP einbinden* beschrieben, ein rudimentäres PHP-Gerüst zum Testen der folgenden Beispiele haben wir in Listing 1 dargestellt.

Listing 1. Der SOAP-Client als Grundgerüst

```
1  <?php
2
3    if ( ! $client = new SOAPClient(
         "https://lsf-server/quisserver/services/soapsearch?WSDL" ) )
5    {
         die( "not connected" );
7    }
     $xmlParams = "[...]";
9    $xmlResult = $client->search( $xmlParams );
     print( $xmlResult );
11
     ?>
```

Nachdem ein neuer SOAP-Client mit der WSDL-Datei des entsprechenden HIS-LSF-Dienstes als Parameter erzeugt wurde (Zeile 3-4), kann eine Abfrage formuliert (Zeile 8, vgl. Abschnitt 4) und damit eine Funktion des Dienstes (in diesem Fall search, Zeile 9) aufgerufen werden.

Die Services können unter der URL %SERVER_NAME/quisserver/services/ abgerufen werden. Dazu sollte immer das sichere Protokoll HTTPS verwendet werden, damit die Kommunikation zwischen LSF Instanz und aufrufender Anwendung über das Netz verschlüsselt wird. In der Definition der über SOAP abrufbaren Objekte in den Konfigurationsdateien der Dienste kann optional eine Authentifizierung verlangt werden, die in der XML-Struktur der Anfrage als Unterelemente $username$, $password$ und/oder $token$ mit übergeben werden müssen.

Generell bietet es sich an, in den aufrufenden Anwendungen – wo immer möglich – Caching-Mechanismen zu nutzen, welche die Ergebnisse der angefragten LSF-Dienste für eine bestimmte Zeit speichern können. So brauchen zum Beispiel die Übungen zu einer Veranstaltungen nicht jedes mal neu über den Dienst anfragt werden, sondern es kann eine einmal angefragte Liste für einen festgelegten Zeitraum wiederverwendet werden. Ist der Zeitraum abgelaufen, wird neu über den Dienst angefragt und der Cache aktualisiert. Dies entlastet die HIS LSF-Instanz, das Netzwerk und bringt meist einen deutlichen Performance-Gewinn in der aufrufenden Anwendung.

Wie schon im vorherigen Abschnitt erwähnt, kann der Dienst *soapsearch* genutzt werden, um Objekte im LSF Datenbestand zu suchen. Hierüber kann bspw. der Schlüssel des Objektes ermittelt werden, der für weitere Abfragen zu den Details des nun bekannten Objektes über den Dienst *dbinterface* verwendet werden kann. Dazu muss zunächst einmal die Anfrage als XML-Struktur formuliert werden (Listing 1, Zeile 8). Neben den o.g. Parametern zur Authentifizierung beschreibt der Parameter *general* das in der Konfigurationsdatei definierte Objekt, das abgefragt werden soll, sowie seine ID zur eindeutigen Identifikation. Diese beiden Unterelemente für Detailabfragen obligatorisch. Optional

Listing 2. Unterstrukturen einer bestimmten Veranstaltung eines bestimmten Semesters

```
<search>
2        <object>veranstaltung</object>
         <language>de</language>
4        <expression>
                 <column name="veranstaltung.veranstnr" value="175101" />
6                <column name="veranstaltung.semester" value="20062" />
         </expression>
8    </search>
```

Dieser Parameter fragt am Objekt veranstaltung *über die Methode* search *des Dienstes* soapsearch *die Unterstrukturen der Veranstaltung 175101 im Semester WS06/07 ab.*

sind hingegen Parameter zur Beschränkung der Auswahl namens *filter*[3] und *condition*. Im Gegensatz zu einer filter-Regel nimmt eine condition-Regel automatisch eine Verknüpfung zwischen den in die Abfrage einbezogenen Tabellen vor. Achtung: Nutzt man den Dienst *soapsearch*, werden die Regeln im Parameter *expression* definiert (vgl. Listing 2, Zeile 4-7). Weitere Parameter legen bei mehrsprachigen Inhalten die Sprache fest, in welcher das Ergebnis erwartet wird (*language*) sowie die Anzahl der zurückgegebenen Datensätze (*limit* und *offset*). Beispiele für Anfragen finden sich im nächsten Abschnitt.

4. Beispiele aus der Praxis

In diesem Abschnitt werden beispielhaft Szenarien konstruiert, um die Handhabung der Schnittstelle besser zu verdeutlichen. Da HIS-LSF über diese Schnittstelle angeschlossene Lernplattformen nicht ereignisbasiert über aktuelle Änderungen informieren kann[4], sind nur diejenigen Szenarien praxisrelevant, die durch das *Pull-Prinzip* (also: die Lernplattformen holen sich diese Informationen) sinnvoll implementiert werden können, ohne dass ein zu großer Overhead entsteht. Diese Entscheidung ist individuell zu treffen und hängt von der einzusetzenden Lernplattform und vom zu unterstützenden Prozess ab. Die Ziffern in Klammern beziehen sich auf die resultierenden Schnittstellenfunktionen in den darauf folgenden Abschnitten.

Szenario 1: Abfrage der Zugehörigkeit zu Veranstaltungen Beim Einloggen eines Studenten in die Lernplattform soll abgefragt werden, in welchen Veranstaltungen/Gruppen der Student lt. HIS LSF Teilnehmer ist (vgl. 4.4.), damit ihm der Zugriff auf die entsprechenden Materialien gewährt werden kann.

Szenario 2: Veranstaltungsseite für Teilnehmer und Interessierte Die Seite einer bestimmten Veranstaltung in der Lernplattform enthält

- **Detailinformationen**, z. B. den Titel und die Veranstaltungsnummer, SWS, Termine, Parallelgruppen, usw. Um an diese Informationen jeweils aktuell zu gelangen, muss die Lernplattform sie aus dem LSF abfragen können (vgl. 4.2.). Da die Lernplattform die LSF-Schlüssel (*veranstid*) noch nicht kennt, muss eine Suche nach den LSF-Schlüsseln der in der Lernplattform bekannten Attribute (Semester, Veranstaltungsnummer) möglich sein (vgl. 4.1.).
- Die hierarchische **Struktur der Veranstaltung**, z. B. Vorlesung, Zentralübung, Übung (letzterer Punkt wiederum unterteilt nach verschiedenen Übungsgruppen). Für jede Ebene innerhalb dieser Struktur gibt es einen eigenen *virtuellen Lernraum*. Beispielsweise enthält der Lernraum zur Übungsgruppe 5 spezielle Informationen des Tutors dieser Gruppe sowie den Abgaberaum für die Übungszettel enthält.

Der Zugriff auf beliebige Elemente in einem Lernraum kann vom Dozenten auf die Teilnehmer der Veranstaltung beschränkt werden (z. B. wg. Urheberrecht). Sofern der Zugriff nur auf die offiziellen Teilnehmer beschränkt ist, können nur Studierende zugreifen, deren Belegstatus im LSF für die Veranstaltung *ZU* (zugelassen) lautet (vgl. 4.4.).

Szenario 3: Veranstaltungs-Management für Dozenten/Vetreter/Tutoren/Mitarbeiter Auf dieser Seite kann ein Dozent (oder ein von ihm ernannter Vertreter/Tutor/Mitarbeiter) seine Veranstaltungen in der Lernplattform auf folgende Weise administrieren:

- Anzeige der Teilnehmer der Gesamtveranstaltung bzw. der Gruppen (vgl. 4.3.) oder Export von Teilnehmerlisten (z. B. im Excel-Format).
- Rundmail-Funktion (vgl. 4.3.). Es kann eine Rundmail an alle Veranstaltungsteilnehmer oder an alle Teilnehmer einer bestimmten Übungsgruppe verschickt werden.

[3] Hiermit können Selektionen eingeschränkt werden. Eine genaue Kenntnis der Datenbankstrukturen sind aber erforderlich.

[4] z. B. die Einrichtung einer neuen Veranstaltung, die Belegung einer Veranstaltung durch einen Studierenden, eine aktuelle Terminverschiebung einer Veranstaltung. Eine automatisierte Systembenachrichtigung bedingt einen Nachrichtenbus und eine ereignisgesteuerte Middleware, wie sie beispielsweise im Konzept der CampusSourceEngine berücksichtigt wird (vgl. [S+04]).

- ❏ Upload von Veranstaltungs-Materialien, Steuerung der Zugriffsrechte darauf
- ❏ Aktuelle Mitteilungen einstellen/ändern/löschen
- ❏ Ernennung von Vertretern, Tutoren und Mitarbeitern durch den Dozenten einer Veranstaltung.

Szenario 4: Individuellen Kalender anzeigen Im individuellen Kalender eines Studenten/Dozenten werden automatisch alle seine Veranstaltungstermine angezeigt. Auch hierzu ist eine Abfrage aller Veranstaltungen eines Studenten im LSF nötig (vgl. 4.4.). Der Student kann zudem manuell weitere Termine zu seinem Kalender hinzufügen.

Im Folgenden werden die aus den obigen Ausführungen resultierenden vier Schnittstellenfunktionen beschrieben und jeweils eine mögliche Umsetzung präsentiert.

4.1. Abfrage aller LSF-Veranstaltungsschlüssel einer konkreten Veranstaltung

Anhand von einem Veranstaltungsschlüssel und einem Semester sollen die Schlüssel aller Unterveranstaltungen ermittelt werden. Zum Beispiel: Die Veranstaltung *Modellierung* (Veranstaltungsnummer 175101, WS06/07) hat die folgenden Veranstaltungsschlüssel:

- 25167 (für die *Vorlesung* Modellierung WS06/07)
- 25168 (für die *Zentralübung* Modellierung WS06/07)
- 25332 (für die *Übung* Modellierung WS06/07)

Dies ist eine typische Suche nach Objekten, die über den Dienst *soapsearch* durchgeführt werden kann. Dazu wird der in Listing 2 beschriebene XML-Code als Parameter der Methode *search* übergeben. Als Ergebnis werden die zu dieser Veranstaltung zugehörigen Unterstrukturen geliefert. Das übergebene Objekt *veranstaltung* ist als Mapping zwischen dem Klassennamen und der entsprechenden Konfigurationsdatei des change-Modules (change/veranstaltung/veranstaltung.xml) in der Konfigurationsdatei soapsearch.xml vorhanden (vgl. Anhang A).

4.2. Abfrage der Veranstaltungsdaten zu einem konkreten Veranstaltungsschlüssel

Im Unterschied zu der beschriebenen Suche in 4.1. ist das Objekt, zu dem Daten geliefert werden sollen, bekannt. Hier kann daher die Methode *getDataXML* des Dienstes *dbinterface* genutzt werden. Der XML-Parameter, der hierzu genutzt werden kann, ist in Listing 3 beschrieben.

Die Methode getDataXML *des Dienstes* dbinterface *liefert mit diesem Parameter die Details zu dem Objekt* Lecture*, das durch die ID 5678 definiert ist.*

Listing 3. Veranstaltungsdetails

```
<SOAPDataService>
    <general>
        <object>Lecture</object>
    </general>
    <id>5678</id>
    <language>de–DE</language>
</SOAPDataService>
```

Die Datenbankabfrage, die durch diese Methode ausgelöst wird, ist in der Konfigurationsdatei soapgetdata.xml hinterlegt (vgl. Anhang B). Es werden folgende Daten zurückgegeben: Veranstaltungsnummer, Semester, Veranstaltungsart (Vorlesung, Seminar, etc.), Titel der Veranstaltung, Anzahl SWS, Textfeld ”Bemerkung” sowie alle Termine zu dieser Veranstaltung.

4.3. Abfrage der Belegungen einer konkreten Veranstaltung

Mit dieser Abfrage soll eine Belegungsliste anhand einer Veranstaltungs-ID erstellt werden. Die Rückgabe der Methode *getDataXML*, die mit dem in Listing 4 beschriebenen Parameter aufgerufen wird, ist eine Liste aller Studenten, die gemäß der LSF-Tabelle *r_belegt* die entsprechende Veranstaltung belegt haben. Jeder Eintrag dieser Liste umfasst jeweils Matrikelnummer, Parallelgruppen-Nr., Belegstatus und Priorität. Die Beschreibung des Objektes CourseDetail, die hierbei genutzt wird, steht im Anhang C.

Der Grund, weshalb die Belegungen nicht einfach unter den Veranstaltungsdaten (s. 4.2.) mitgeliefert werden sollen, ist der teilweise recht große Umfang: Teilnehmerzahlen um die 500 Studierende sind in manchen Studiengängen keine Seltenheit. Schließlich ist für viele Abfragen der reinen Veranstaltungsdaten die Belegungsliste gar nicht relevant.

Listing 4. Belegungsliste einer Veranstaltung

```
<SOAPDataService>
        <general>
                <object>CourseDetail</object>
        </general>
        <id>5678</id>
        <language>de-DE</language>
</SOAPDataService>
```

Mit diesem XML-Parameter aufgerufen liefert die Methode getDataXML *des Dienstes* dbinterface *die Belegliste der Veranstaltung mit der ID 5678.*

4.4. Abfrage aller Belegungen eines konkreten Studenten

In dieser letzten Anforderung sollen die belegten Veranstaltungen eines Studierenden für ein gegebenes Semester (gemäß LSF-Tabelle *r_belegt* zurückgegeben werden. Jeder Eintrag dieser Liste umfasst jeweils Veranstaltungsschlüssel, Parallelgruppen-Nr., Belegstatus und Priorität. Der entsprechende XML-Parameter für die Methode *getDataXML* für den Dienst *dbinterface* ist in Listing 5 beschrieben, die entsprechende XML-Struktur für die Konfiguration der Schnittstelle in Anhang D.

Listing 5. Gebuchte Veranstaltungen eines Studierenden

```
<SOAPDataService>
        <general>
                <object>CourseListAll4Student</object>
        </general>
        <mtknr>1234567</mtknr>
        <semester>20062</semester>
        <language>de-DE</language>
</SOAPDataService>
```

Mit diesem XML-Parameter aufgerufen liefert die Methode getDataXML *des Dienstes* dbinterface *die Belegliste der Veranstaltung*

5. Resümee und Ausblick

Ein Qualitätskriterium von Web-Services ist die Kapselung der ihnen zugrunde liegenden Funktionalität und Komplexität, sowie die Bereitstellung einer einfachen Programmierschnittstelle. Hiervon ist die HIS-SOAP-Schnittstelle sehr weit entfernt. Die in diesem Papier beschriebenen HIS-LSF-Dienste bieten nur Konstrukte an, mit denen Datenbankabfragen abgesetzt und deren Ergebnisse übermittelt werden können. Die eigentliche Funktionalität/Semantik muss jedoch für die einzelnen Universitäten durch die Definition dieser Anfragen und Ergebnisse konfiguriert werden. Aus diesem Blickwinkel kann man also nicht von einer Standardschnittstelle zum HIS-LSF sprechen, die für alle Universitäten mit der gleichen Semantik genutzt werden kann. Hierbei wurde großes Potenzial verschenkt, zur Schnittstellenkonsolidierung universitärer Systeme beizutragen.

Auf der anderen Seite bieten die vorgestellten Web-Dienste jegliche Freiheiten, die man sich wünschen kann. Durch die individuellen Konfigurationsmöglichkeiten lassen sich Anfragen für alle Datenbanken definieren, die über das PublishModul erreicht werden können[5]. Dies erfordert allerdings eine gewisse Einarbeitungszeit und wird durch die an vielen Stellen nur unzureichende Dokumentation erschwert. Gefragt sind dazu nicht nur Programmierfähigkeiten in XML, SOAP und der Sprache der anfragenden Anwendung, sondern auch eine genaue Kenntnis der Datenbanken zur Definition der für die Konfiguration notwendigen (SQL-)Abfragen.

[5] Aus technologischer Sicht lässt sich sogar ohne Probleme ein schreibender Zugriff realisieren, wenn dieser benötigt werden würde.

Was in diesem Papier bereits an mehreren Stellen anklang, soll hier noch einmal bewusst angemerkt werden: Die untersuchte Schnittstelle ist nicht in der Lage, eine Nachrichten-basierte, Ereignis-gesteuerte Kommunikation mit Lernplattformen zu implementieren, wie sie für eine alltagstaugliche Lösung an Universitäten benötigt wird (vgl. [RHG05] für ein solches Szenario). Es können also nur Szenarien abgebildet werden, die auf dem so genannten *Pull-Mechanismus*, also dem aktiven Abrufen der Daten aus den angebundenen Lernplattformen, basieren. Ein *Push-Mechanismus*, also das passive "im Falle eines Ereignisses benachrichtigt werden", ist mit dem des HIS-LSF zugrunde liegenden Architekturkonzept nicht möglich.

Für eine alltagstaugliche Schnittstelle muss also das Architekturkonzept der HIS-Software grundlegend geändert werden. Die von der CampusSource-Initiative vorangetriebene CampusSourceEngine (CSE, vgl. [S$^+$04]), welche HIS-LSF an einen solchen Nachrichtenbus anbinden würde, hat derzeit leider immer noch keine Produktreife[6]. Außerdem hat sich die HIS GmbH diesbezüglich immer noch nicht klar positioniert. Wird auf medienwirksamen Terminen regelmäßig der neuste Stand der CampusSourceEngine präsentiert, existiert bis dato immer noch kein offizieller Kooperationsvertrag zwischen HIS und CampusSource.

Die HIS hat im Dezember 2006 eine neue Generation ihrer Software angekündigt mit dem Namen *HISinOne*, die vollständig auf einer Service-orientierten Architektur basieren soll. Ob allerdings auch ein Nachrichten-basierter, Ereignis-gesteuerter Austausch zwischen Komponenten möglich ist, lässt sich den Foliensätzen vom Auftaktworkshop *Von Hochschulen für die Hochschulen*[7] vom 14.12.2006 leider nicht entnehmen.

Literatur

KERRES, M. ; NATTLAND, A. ; WECKMANN, H.-D.: Hybride Lernplattformen und integriertes Informationsmanagement an der Hochschule. In: DITTRICH, K. (Hrsg.) ; Gesellschaft für Informatik (Veranst.): *Informatik 2003, Innovative Informatikanwendungen* Bd. 2 Gesellschaft für Informatik, 2003, S. 90–96

ROTH, A. ; HOPPE, G.: Approaching Heterogeneity within Educational Technology - Concepts, Risks, and Success Factors of Launching Service-oriented Architectures. In: PEARSON, E. (Hrsg.) ; BOHMAN, P. (Hrsg.) ; AACE (Veranst.): *Proceedings of Ed-Media 2006*. Orlando/Florida, 2006, S. 2228–2233

ROTH, P. ; HINZ, D. ; GAST, Ch.: Integration von Learning Management in bestehende IT-Infrastrukturen. In: *Information Management & Consulting* 20 (2005), Nr. 1, S. 69–76

ROTH, A. ; SCHOLZ, M. ; SUHL, L.: Webbasiertes Lehrveranstaltungsmanagement – Effizienzsteigerung durch horizontale Integration von Lehr-/Lerntechnologien. In: CARSTENSEN, D. (Hrsg.) ; BARRIOS, B. (Hrsg.) ; Gesellschaft für Medien in der Wissenschaft (Veranst.): *Campus 2004 – Kommen die digitalen Medien an den Hochschulen in die Jahre?* Bd. 29. Münster : Waxmann Verlag, 2004, S. 438–449

STEPPING, M. u. a.: CampusSourceEngine – Die Schnittstelle von e-Learning Systemem zum HIS-GX System der HIS GmbH / CampusSource. Hagen, Hannover, 2004. – Forschungsbericht

[6] Derzeit wird es an der Fachhochschule für Wirtschaft Berlin genutzt, um die im HIS-LSF verwalteten Veranstaltungsteilnehmer mit einer Lernplattform synchron zu halten. Darüber hinausgehende Prozesse wie bspw. Terminänderungen sind nach unserem Informationsstand derzeit noch nicht umgesetzt.

[7] Siehe hierzu http://www.hisinone.de.

A Mapping der Lehrveranstaltung (Auszug aus soapsearch.xml)

Das in Listing 2 genutzte Veranstaltungsobjekt ist als Mapping in der Konfigurationsdatei soapsearch.xml definiert.

```
<object name="veranstaltung">
        <search-form subdir="veranstaltung" moduleParameter="veranstaltungSearch" />
        <search-result subdir="veranstaltung" moduleParameter="veranstaltung" />
</object>
```

B Details zu einer Lehrveranstaltung (Auszug aus soapgetdata.xml)

```
<Lecture RootName="veranstaltung" extDataAccess="y" context="admin" database="lsf"
        statistics="small" conditionTables="veranstaltung" authentication="y">
        <SOAPRelation>
                <id>
                        <veranstaltung.veranstid/>
                </id>
        </SOAPRelation>
<!-- Basisangaben zur Veranstaltung -->
        <Transform.SQL ElementName="veranstaltung" useLangID="id" emptySet="n">
                SELECT veranstaltung.veranstid AS id, veranstaltung.veranstnr As veranstnr,
                veranstaltung.semester AS semester, k_verart.dtxt AS typ, veranstaltung.dtxt AS titel,
                veranstaltung.ktxt AS kurzbeschreibung, veranstaltung.sws AS sws
                FROM veranstaltung, k_verart [from]
                WHERE veranstaltung.veranstid=[id]
                AND veranstaltung.verartid=k_verart.verartid
                <!--AND (k_verart.kategorie IN ('L', 'J') OR k_verart.kategorie IS NULL)-->
                [where]
                <Transform.Blob ElementName="bemerkung" table="veranstaltung" column="bemerkung"
                        checkID="id"/>
                <Transform.Parent ElementName="termine"><Transform.SubSQL ElementName="termin"
                        ignoreFields="TerID, TerRaum" emptySet="n">
                                SELECT veransttermin.vtid AS TerID, veransttermin.parallelid AS parallelid,
                                k_wochentag.ktxt AS wochentag, veransttermin.beginn AS start,
                                veransttermin.ende AS ende, k_rhythmus.ktxt AS rhythmus,
                                veransttermin.rgid AS raum , veransttermin.begindat AS startdatum,
                                veransttermin.endedat AS enddatum
                                FROM veransttermin, k_wochentag, k_rhythmus
                                WHERE veransttermin.tabpk=[id]
                                AND veransttermin.tabelle='veranstaltung'
                                AND veransttermin.wochentagid=k_wochentag.wochentagid
                                AND k_rhythmus.rhythmusid=veransttermin.rhythmusid
                                ORDER BY k_wochentag.sort, veransttermin.beginn, veransttermin.ende
                                <!-- Räume zum Termin -->
                                <Transform.SubSQL ElementName="raum" emptySet="n">
                                        SELECT raum.rgid AS id, raum.dtxt AS "text"
                                        FROM raum
                                        WHERE raum.rgid=[TerRaum]</Transform.SubSQL>
                                <!-- Dozenten -->
                                <Transform.SubSQL ElementName="person" emptySet="n">
                                        SELECT personal.pid AS id, personal.nachname AS "text"
                                        FROM r_veranstpers, personal
                                        WHERE r_veranstpers.pid=personal.pid
                                        AND r_veranstpers.vtid=[TerID]
                                        ORDER BY r_veranstpers.sort, personal.nachname</Transform.SubSQL>
                                <!-- Personengruppen -->
                                <Transform.SubSQL ElementName="persongroup" emptySet="n">
                                        SELECT k_persgroup.persgroupid AS id, k_persgroup.dtxt As "text"
                                        FROM k_persgroup, r_persgroup
                                        WHERE r_persgroup.persgroupid = k_persgroup.persgroupid
                                        AND r_persgroup.vtid = [TerID]
                                        ORDER BY r_persgroup.sort</Transform.SubSQL>
                        </Transform.SubSQL></Transform.Parent>

                <Transform.Parent ElementName="zuordnung"><Transform.SubSQL
                        ElementName="studiengang" emptySet="n">
                                SELECT r_zuordabstgv.kommentar AS kommentar, r_zuordabstgv.credits
```

```
                                AS credits, k_abstgv.ktxt AS abschluss, k_bereich.ktxt AS bereich, k_teilgeb.ktxt
58                              AS teilgeb, k_stufe.ktxt AS stufe, k_phase.ktxt AS phase, k_modul.ktxt AS modul
                                FROM veranstaltung, r_zuordabstgv, k_abstgv, k_bereich, k_teilgeb, k_phase,
60                              k_stufe, k_modul
                                WHERE veranstaltung.veranstid = [id]
62                              AND r_zuordabstgv.veranstid = veranstaltung.veranstid
                                AND r_zuordabstgv.abstgvnr = k_abstgv.abstgvnr
64                              AND ((r_zuordabstgv.bereichid = k_bereich.bereichid)
                                OR (r_zuordabstgv.bereichid IS NULL))
66                              AND ((r_zuordabstgv.teilgebid = k_teilgeb.teilgebid)
                                OR (r_zuordabstgv.teilgebid IS NULL))
68                              AND ((r_zuordabstgv.stufeid = k_stufe.stufeid)
                                OR (r_zuordabstgv.stufeid IS NULL))
70                              AND ((r_zuordabstgv.phaseid = k_phase.phaseid)
                                OR (r_zuordabstgv.phaseid IS NULL))
72                              AND (r_zuordabstgv.modulid = k_modul.modul
                                OR r_zuordabstgv.modulid IS NULL)
74                      </Transform.SubSQL></Transform.Parent>
        </Transform.SQL>
76 </Lecture>
```

C Belegungen einer Lehrveranstaltung (Auszug aus soapgetdata.xml)

```
 <CourseDetail RootName="belegungsliste" extDataAccess="y" context="admin" database="lsf"
2       statistics="small" conditionTables="veranstaltung" authentication="y">
        <SOAPRelation>
4               <id>
                        <veranstaltung.veranstid/>
6               </id>
        </SOAPRelation>
8       <Transform.SQL ElementName="veranstaltung" database="lsf" useLangID="id" emptySet="n">
                SELECT veranstaltung.veranstid AS veranstid,
10              veranstaltung.dtxt AS veranstname,
                veranstaltung.ltxt AS description
12              FROM veranstaltung [from]
                WHERE veranstaltung.veranstid=[id] [where]
14              <Transform.SubSQL ElementName="student" database="lsf" emptySet="n">
                        SELECT r_beleg.tabpk AS mtknr, r_beleg.parallelid AS parallelid,
16                      r_beleg.status AS status, r_beleg.prioritaet AS prioritaet FROM r_beleg
                        WHERE r_beleg.veranstid=[id]
18              </Transform.SubSQL>
        </Transform.SQL>
20 </CourseDetail>
```

D Belegungsliste eines Studenten für ein Semester (Auszug aus soapgetdata.xml)

```
 <CourseListAll4Student RootName="belegteVeranstaltungen" extDataAccess="y" context="admin"
2 database="lsf" statistics="small" conditionTables="veranstaltung, r_beleg" authentication="y">
        <SOAPRelation>
4               <mtknr>
                        <r_beleg.tabpk/>
6               </mtknr>
                <semester>
8                       <veranstaltung.semester/>
                </semester>
10      </SOAPRelation>
        <Transform.SQL ElementName="student" database="lsf" useLangID="id" emptySet="n">
12              SELECT DISTINCT r_beleg.tabpk AS mtknr FROM r_beleg [from] WHERE
                r_beleg.tabpk = [mtknr] [where]
14              <Transform.Parent ElementName="veranstaltungen"><Transform.SubSQL
                ElementName="veranstaltung" database="lsf" useLangID="id" emptySet="n">
16                              SELECT r_beleg.veranstid AS veranstid,
                                veranstaltung.veranstnr AS veranstnr,
18                              veranstaltung.dtxt AS veranstname,
                                r_beleg.parallelid AS parallelid,
```

```
20                                  r_beleg.status AS status,
                                    r_beleg.prioritaet AS prioritaet
22                                  FROM r_beleg, veranstaltung
                                    WHERE r_beleg.veranstid = veranstaltung.veranstid
24                                  <!--AND r_beleg.status='ZU' -->
                                    AND r_beleg.tabpk = [mtknr]
26                                  AND veranstaltung.semester = [semester]
                                    ORDER BY r_beleg.veranstid
28
                            </Transform.SubSQL></Transform.Parent>
30          </Transform.SQL>
    </CourseListAll4Student>
```