Open Library WebServices
Technische Dokumentation

Oliver Flimm <flimm@openbib.de>
Stand: 30. März 2005

Inhaltsverzeichnis

1 Hintergrund und Zielsetzung

Ein großes Problem in der derzeitigen Bibliothekslandschaft ist die Zersplitterung der angebotenen Dienste. Typischerweise bildet ein kommerzielles Bibliothekssystem mit Diensten wie Katalogisierung, Erwerbung, Ausleihe und Recherche über einen WebOPAC die Basis, an die andere, meist externe Dienste wie zusätzliche Datenbankrecherchen oder Ähnliches angebunden werden müssen. Ebenso besteht die Möglichkeit, dass die Dienste des eigentlichen Bibliothekssystems nicht vollständig integriert sind und auch hier eine gegenseitige Anbindung erst geschaffen werden muss.

Als Auswege werden u.a. monolithische Portallösungen der jeweiligen Anbieter von Bibliothekssystemen gesehen, da diese naturgemäß die Anbindung des wichtigen Dienstes Ausleihe mitbringen. Die Anbindung dieses Dienstes kann aufgrund seiner generellen Wichtigkeit dann als wesentlicher angesehen werden, als die Qualität und die Fähigkeiten des jeweiligen Portals.

Sinnvoller und mit deutlich mehr Freiheit bei der Auswahl des 'besten Systems für den jeweiligen Zweck' verbunden ist die Kopplung unterschiedlicher Systeme über geeignete Kommunikationsschnittstellen herzustellen.

Ein international standardisiertes Verfahren, diese Kommunikationsschnittstelle zu verwirklichen sind WebServices auf der Basis von XML-RPC oder SOAP. Damit können verschiedene Programmiersprachen verwendet werden.

Eine wesentliche Voraussetzung für die Integrationsfähigkeit ist die Offenheit der Schnittstelle. Nur damit ist die Anbindung beliebiger Systeme gewährleistet, während man bei der Verwendung proprietärer Protokolle den Interessen des jeweiligen Unternehmens ausgeliefert ist.

Ein weiterer Vorteil einer offenen Schnittstelle ist die langfristige Nutzung von Anwendungen, die auf Grundlage dieser Schnittstelle programmiert wurden, auch bei möglichen Systemwechseln. Lediglich die Realisierung der Webservices für das jeweilige neue Bibliothekssystem ist dann zu ändern.

Aus diesem 'Freiheitsgedanken' heraus und der entsprechenden konkreten Problemstellung im OpenBib Recherche-Portal, bei der genau diese Fragen bei der Anbindung des lokalen Bibliothekssystems eine Rolle spielten, wurde mit der Implementierung einer freien, auf SOAP basierenden WebServices-Schnittstelle begonnen.

Die Open Library WebServices sind geboren.

Wie schon OpenBib wurden Sie ursprünglich als Freizeitprojekt unter der GPL als OpenSource-Software begonnen und werden derzeit im Wesentlichen auch in diesem Rahmen weiterentwickelt. Geplant ist jedoch die Ausweitung auf weitere interessierte Entwickler, um eine allgemeine Nutzung der Schnittstelle insbesondere im Hinblick auf zusätzliche Funktionalitäten weiter voranzutreiben.

Insofern ist jeder herzlich eingeladen, bei der Weiterentwicklung und Definition der Schnittstelle mitzuwirken.

2 Referenz

2.1 Implementierung

Die Open Library WebServices, kurz OLWS, wurden mit dem Perl-Modul `SOAP::Lite` entwickelt. Damit ist eine Implementierung weiterer Funktionen sehr einfach möglich. Durch die Verwendung von SOAP kann jedoch prinzipiell auch jede andere Programmiersprache genutzt werden, um die WebServices anzusprechen bzw. die bestehenden WebServices zu reimplementieren.

2.2 WebServices

2.2.1 urn:/Authentication

Beschreibung

Über dieses WebService wird die Authentifizierung am jeweiligen System gesteuert.

Unterstützte Zielsysteme	`Sisis`
Implementiert über die Module	`OLWS::Sisis::Authentication.pm`

Funktionen

Funktionsname	`authenticate_user`
Argumente	(`Benutzername, Passwort, Datenbankname`)
Rückgabe bei Erfolg	Benutzerinformationen: Vorname, Nachname, Geschlecht, Geburtsdatum, Ort, Strasse, PLZ, Soll, Guthaben, Avanz, Branz, Bestellanz, Pflanz, Vmanz, Maanz, Vlanz, Sperre, Sperrdatum, Email

2.2.2 urn:/Circulation

Beschreibung

Über dieses WebService werden benutzerkonto- bzw. ausleihspezifische Funktionen gesteuert.

Unterstützte Zielsysteme	`Sisis`
Implementiert über die Module	`OLWS::Sisis::Circulation.pm`

Funktionen

Name	`get_borrows`
Beschreibung	Lieferung von Ausleihinformationen zu einem authentifizierten Nutzer einer spezifischen Zieldatenbank
Argumente	`(Benutzername, Passwort, Datenbankname)`
Rückgabe bei Erfolg	Array mit Ausleihinformationen: Katkey, Signatur, MTyp, Mediennummer, AusleihDatum, RueckgabeDatum

Name	`get_orders`
Beschreibung	Lieferung von Bestellinformationen zu einem authentifizierten Nutzer einer spezifischen Zieldatenbank
Argumente	`(Benutzername, Passwort, Datenbankname)`
Rückgabe bei Erfolg	Array mit Bestellinformationen: Katkey, Signatur, MTyp, Status, Mediennummer, BestellDatum

Name	`get_reservations`
Beschreibung	Lieferung von Vormerkungsinformationen zu einem authentifizierten Nutzer einer spezifischen Zieldatenbank
Argumente	`(Benutzername, Passwort, Datenbankname)`
Rückgabe bei Erfolg	Array mit Vormerkungsinformatione: Katkey, Mediennummer, VormerkDatum, AufrechtDatum, Stelle

Name	`get_reminders`
Beschreibung	Lieferung von Überziehungs- bzw. Gebühreninformationen zu einem authentifizierten Nutzer einer spezifischen Zieldatenbank
Argumente	`(Benutzername, Passwort, Datenbankname)`
Rückgabe bei Erfolg	Array mit Überziehungsinformationen: Katkey, Mediennummer, AusleihDatum, Leihfristende, Mahngebuehr, Saeumnisgebuehr

2.2.3 urn:/Media

Beschreibung

Über dieses WebService wird der Zugriff auf die Katalogdaten im Zielsystem gesteuert.

Unterstützte Zielsysteme	`Sisis`
Implementiert über die Module	`OLWS::Sisis::Media.pm`

Funktionen

Name	`get_native_title_by_katkey`
Beschreibung	Lieferung der Titelinformationen zu einem Katkey direkt aus den nativen Datenbank-Blobs
Argumente	`(Datenbankname, Katkey)`
Rückgabe bei Erfolg	Titeldaten im jeweiligen nativen Kategorienschema

2.2.4 urn:/MediaStatus

Beschreibung

Über dieses WebService wird der Medienausleihstatus zu den jeweiligen Medien geliefert.

Unterstützte Zielsysteme	`Sisis`
Implementiert über die Module	`OLWS::Sisis::MediaStatus.pm`

Funktionen

Funktionsname	`get_mediastatus`
Argumente	`(Katkey, Datenbankname)`
Rückgabe bei Erfolg	Ausleihstatus eines Mediums Signatur, Exemplar, Standort, Status, Rueckgabe

3 Programmierbeispiele

Für die Programmierung der WebServices in Perl wird das Modul `SOAP::Lite` genutzt. Mit diesem ist es sehr einfach möglich sowohl die Client- wie auch die Server-Seite der SOAP-Kommunikationspartner zu implementieren.

3.1 Client

Der Client wird durch folgendes Grund-Konstrukt implementiert. Beispielhaft wird der Programm-Code der Funktion `authenticate_olws_user` aus `OpenBib::Login::Util` verwendet.

```
use SOAP::Lite;
```

Mit `use` wird das Modul `SOAP::Lite` importiert.

```
sub authenticate_olws_user {
[...]

  my $soap = SOAP::Lite
  -> uri("urn:/Authentication")
  -> proxy($circcheckurl);
```

Es wird ein SOAP-Objekt `$soap` erzeugt, welches den Service `urn:/Authentication` anspricht, der über den effektiven URL `$circcheckurl` erreicht wird. Hinter `$circcheckurl` kann sich serverseitig ein `mod_perl`-Handler, ein CGI-Skript oder ein eigenständiger Dämon verstecken.

```
  my $result = $soap->authenticate_user($username,$pin,$circdb);
```

Über das SOAP-Objekt `$soap` wird auf dem entfernten Server die Methode `authenticate_user` mit entsprechenden Argumenten aufgerufen. Das Ergebnis dieses Aufrufs wird in `$result` abgespeichert.

```
  my %userinfo=();

  unless ($result->fault) {

    if (defined $result->result){
```

```
      %userinfo = %{$result->result};
      $userinfo{'erfolgreich'}="1";
    }
    else {
      $userinfo{'erfolgreich'}="0";
    }
  }
  else {
    $logger->error("SOAP Authentication Error", join ', ',
      $result->faultcode, $result->faultstring, $result->faultdetail);
  }
```

Je nachdem, ob der Aufruf erfolgreich war bzw. nicht, werden verschiedene Aktionen durchgeführt, z.B. eine Fehlermeldung via `log4Perl` geloggt.

```
[...]
}
```

3.2 Server

Der Server kann als CGI-Programm, als eigenständiger Dämon oder unter mod_perl laufen. Aus Performance-Gründen sollte auf die CGI-Variante im Produktionsbetrieb verzichtet werden. Für den Testbetrieb ist sie jedoch sehr gut geeignet.

Grundsätzlich besteht der Server aus einem Dispatcher, der als CGI-Skript, Dämon oder mod_perl-Modul läuft. Dieser Dispatcher nimmt die Anfragen entgegen und verteilt sie entsprechend der in ihm konfigurierten Namensräume auf ebenfals konfigurierte Klassen oder Module.

Ein Beispiel für einen einfachen Dispatcher auf CGI-Basis lautet:

```
use SOAP::Transport::HTTP;
use Log::Log4perl;

Log::Log4perl->init_once($OLWS::Config::config{log4perl\_path});

SOAP::Transport::HTTP::CGI
  -> dispatch_with({
                  'urn:/Authentication' => 'OLWS::Sisis::Authentication',
                  'urn:/Circulation'    => 'OLWS::Sisis::Circulation',
                  'urn:/Media'          => 'OLWS::Sisis::Media',
                  'urn:/MediaStatus'    => 'OLWS::Sisis::MediaStatus',
                   })
  -> handle;
```

In diesem Dispatcher werden die Namenräume `urn:/Authentication`, `urn:/Circulation`,

urn:/Media und urn:/MediaStatus definiert sowie diesen Namensräumen die Module OLWS::Sisis::*, in denen die Funktionen implementiert sind, zugeordnet.

Für die Implementation muss dann nur noch das entsprechende Modul OLWS::xyz::* mit seinen Funktionen in Form eines Packages geschrieben werden. So ein Modul hat folgende Form:

```
package OLWS::Sisis::Media;

use strict;
use warnings;

use Log::Log4perl qw(get_logger :levels);
use DBI;
use OLWS::Sisis::Config;
use OLWS::Sisis::Data;

# Importieren der Konfigurationsdaten als Globale Variablen
# in diesem Namespace

use vars qw(%config);

*config=\%OLWS::Sisis::Config::config;

sub get_native_title_by_katkey {

  my ($class, $database, $katkey) = @_;

  # Log4perl logger erzeugen

  my $logger = get_logger();

  ######################################################################
  # Verbindung zur SQL-Datenbank herstellen

  my $dbh=DBI->connect("DBI:$config{dbimodule}:dbname=$database;
                        server=$config{dbserver};
                        host=$config{dbhost};port=$config{dbport}",
                        $config{dbuser}, $config{dbpasswd})
     or $logger->error_die($DBI::errstr);

  my $sikfstabref=OLWS::Sisis::Data::get_sikfstabref($dbh,$database);
  my $titelref=OLWS::Sisis::Data::get_titref_by_katkey($sikfstabref,
                                            $dbh,$database,$katkey);

  return $titelref;
}

1;
```

Im Wesentlichen beschränkt sich die Implementation auf die **Definition einer Funktion**, hier

```
sub get_native_title_by_katkey {
```

mit einer **Parameterliste**, hier

```
  my ($class, $database, $katkey) = @_;
```

wobei `$class` bei nicht objektorientierter Programmierung nicht verwendet wird, einen **Verarbeitungsbereich**, hier

```
 # Log4perl logger erzeugen

  my $logger = get_logger();

  ######################################################################
  # Verbindung zur SQL-Datenbank herstellen

  my $dbh=DBI->connect("DBI:$config{dbimodule}:dbname=$database;
                        server=$config{dbserver};
                        host=$config{dbhost};port=$config{dbport}",
                        $config{dbuser}, $config{dbpasswd})
     or $logger->error_die($DBI::errstr);

  my $sikfstabref=OLWS::Sisis::Data::get_sikfstabref($dbh,$database);
  my $titelref=OLWS::Sisis::Data::get_titref_by_katkey($sikfstabref,
                                                       $dbh,$database,$katkey);
```

in dem sich die eigentliche Programmlogik befindet und schließlich der Rückgabe entsprechender Werte, hier

```
  return $titelref;
```

Hiermit wurde der WebService `get_native_title_by_katkey` mit den Parametern `(Datenbank, Katalogschlüssel)` definiert.

Bei der Verwendung von `die`, oder hier `error_die`, wird die Funktion abgebrochen und Fehlercode/Meldung via SOAP zurückgeliefert.