Probado: WSDL Dokumentation

September 13, 2011

Contents

1 Einleitung **3**

2 RepositoryRegistration **5**
2.1 Operation 5
2.2 Parameter 6
2.2.1 RepositoryRegistrationType 6
2.2.2 RepositoryRegistrationQueryEngineType 6
2.2.3 RepositoryRemoveQueryEngineType 7
2.2.4 RepositorySetAvailabilityType 7

3 RepositoryMetadataExchange **7**
3.1 Operationen 7
3.2 Parameter 7
3.2.1 RepositoryMetadataExchangeType 8

4 Probado Search **9**
4.1 Operationen 10
4.2 Anfrage-Parameter 10
4.2.1 ProbadoContentSearchType 10
4.2.2 ProbadoMetadataSearchType 11
4.3 Antwort-Parameter 11
4.3.1 ProbadoSearchResultType 11

5 Repository Search **13**
5.1 Operationen 13
5.2 Anfrage-Parameter 13
5.3 Antwort-Parameter 13

6 ProbadoFaultType **13**

A Beispiele **14**
A.1 Repository Anmeldung 14
A.1.1 Anfrage 14
A.1.2 Antwort 14
A.2 Query Engine hinzufügen 15
A.2.1 Anfrage 15
A.2.2 Antwort 15
A.3 Query Engine entfernen 15
A.3.1 Anfrage 15
A.3.2 Antwort 16
A.4 Repository Status ändern 16
A.4.1 Anfrage 16
A.4.2 Antwort 17
A.5 Metadata hinzufügen 17
A.5.1 Anfrage 17
A.5.2 Antwort 18
A.6 RepositoryMetadataExchange Client Code 18
A.7 Metadata durchsuchen 19
A.7.1 Anfrage 19
A.7.2 Antwort 20

1 Einleitung

Dieser Report gibt einen Überblick über die verschiedenen Web Services die entweder vom PROBADO-Core-System angeboten oder von diesem verwendet werden.

Die Web Services werden als WSDL Dokument eindeutig spezifiziert. Im wesentlichen besteht ein WSDL Dokument (siehe `http://www.w3schools.com/wsdl/wsdl_documents.asp` aus

- Types: Entsprcht in "'normalen"' Programmiersprachen einer Klasse. So können Übergabeparameter für einen Webservice zusammengefasst werden.
- Messages: Beschreibt welche Typen eine Operation verwendet.
- PortTypes: Definiert die eigentlichen Web Servies (Methoden / Operationen) auf die zugegriffen werden kann. Es muss eine eingehende, ausgehende und fehlerbeschreibende Nachricht zu jedem PortType zugewiesen werden.
- Binding: Regelt Protokoll / Transport der Nachrichten. Wir verwenden *immmer* SOAP XML over HTTP version 1.1[1]

Implementiert werden die Web Services, in dem das WSDL in eine Programmiersprache übersetzt wird. Dies geschieht im Regelfall automatisch, insofern geeignete Tools verfügbar sind. Gehostet wird der Web Service im Regelfall über einen geeigneten Webserver. Hierzu muss abschließend der *Endpoint* (Adresse) definiert werden, unter der der Webservice erreicht werden kann. Dann stehen die Web Services zur Verfügung und können von Clientprogrammen *konsumiert* werden. Dies wiederum geschieht indem man für den Client einen *Proxy* aus der WSDL-Datei ableitet (im Regelfall ebenfalls automatisch), der die Web Services für den Client als gewöhnliche Methoden darstellt, die Ausführung jedoch dem Web Service Host überlässt.

Manuell kann ein Webservice ebenfalls konsumiert werden. Aus der WSDL geht eindeutig das XML Schema der zu verschickenden Nachrichten hervor. Dann wird lediglich ein XML Reader/Writer benötigt, der solche XML Strings baut und ein Webclient der diese XML Strings als Nachrichten von einem Webserver abruft und hinschickt.

Die Web Services definieren die Schnittstellen zwischen den einzelnen Schichten des PROBADO-Systems (siehe 1). Zwischen dem Frontend (Layer 1) und dem Core-System (Layer 2) werden zwei Schnittstellen angeboten:

ProbadoSearch kapselt die Suchfunktionalität des Systems

ProbadoUser stellt die administrativen Funktionen (User-, Sessionverwaltung, ...) zur Verfügung. *Anmerkung: In dieser Version noch nicht vorgesehen.*

[1] `http://schemas.xmlsoap.org/soap/http`

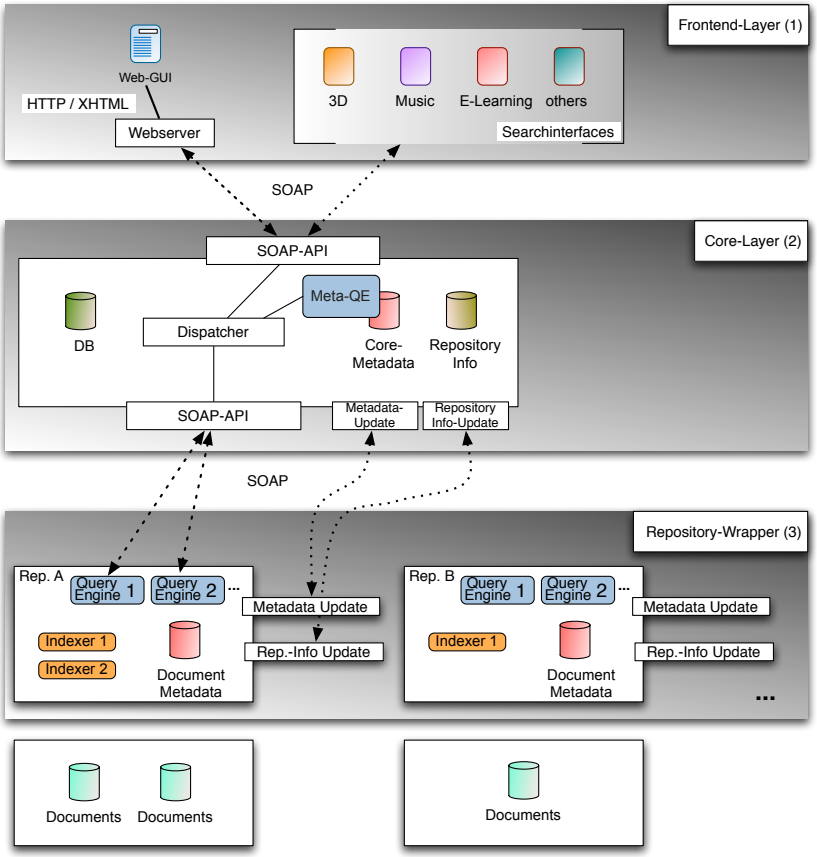

Figure 1: PROBADO Systemarchitektur

Die Suchfunktionalität auf den einzelnen Repositories wird entsprechend von diesen über die Schnittstelle `RepositorySearch` angeboten, auf die der Core zugreift. Aufgabe des Cores ist es, Suchanfragen an alle zuständigen Repositories weiterzuleiten und die Ergebnisse zusammenzufassen.

Weiterhin bietet der Core (L2) noch zwei Schnittstellen für Repositories (L3) an:

RepositoryRegistration Anmeldung von Repositories am Core durch Übermittelung ihrer Systeminfos

RepositoryMetadataExchange Übermittelung der Metadaten aller Dokumente eines Repositories an den Core

Zur Definition der Webservices verwenden wir den "Document/Literal Wrapped Style". Typen, die mehrmals verwendet werden, werden in einer extra XSD

Datei definiert und von den WSDL Definitionen importiert. So erhalten wir keine Redundanz und übersichtlichen, wartbaren Code.

In diesem Dokument werden nun alle Nachrichten *skizziert*, die durch die Webservices ausgetauscht werden. Somit ist ein leicht verständlicher Überblick gewährleistet. Für die tatsächliche Implementierung der Webservices sowie dem konsumieren der Core Webservices sei auf die tatsächlichen WSDL und XSD Dokumente verwiesen.

Die Anfrage- und Antwortparameter der Webservices werden hinsichtlich Datentyp und Häufigkeit genauer spezifiziert. Der Datentyp *complexType* beudeutet, dass sich dieser Parameter aus mehreren Datentypen zusammensetzt. Für Details ist das WSDL zu referenzieren. Neben der Angabe ob ein Paramter optional oder notwendig ist, zeigt *array* an, dass es sich um eine Liste mit Werten dieses Typs handelt.

2 RepositoryRegistration

Der *RepositoryRegistration* Webservice ist Schnittstelle zwischen Repositories (L3) und Core (L2). Der Core hostet diesen Webservice und bietet somit einheitlichen Zugfriff zur Anmeldung von Repositories an. Der Core kann nur an angemeldete Repositories Anfragen weiterleiten. Weiterhin nimmt der Core nur Metadaten von angemeldeten Repositories an (siehe 3). Entsprechend ist das *konsumieren* des RepositoryRegistration Webservice für Repository Betreiber zwingend erforderlich.

2.1 Operation

Dieser Webservice bietet folgende, modularen Methoden. Als Antwort liefern all diese Methoden lediglich *true* oder *false*.

doRepositoryRegistration: Registriert ein Repository mit seinen grundlegenden Systemdaten. Notwendig damit Suchanfragen zu dem Repository weitergeleitet werden können und damit der CORE Metadaten dieses Repositories annimmt.

doRepositoryAddQueryEngine: Optional. Fügt einem bereits registrierten Repository eine Query Engine samt der unterstützten Formate hinzu. Wichtig: Ohne QE wird jede Anfrage (egal was für ein Format) an das Repository weitergeleitet. Wird min. eine QE angegeben, werden nur noch vom Repository (bzw. den eingetragenen QE) unterstützte Query-Formate weitergeleitet.

doRepositoryRemoveQueryEngine: Entfernt eine Query Engine von einem Repository.

doRepositorySetAvailability: Ändert den Verfügbarkeitsstatus eines Repositories.

2.2 Parameter

Nachfolgend werden alle Parameter beschrieben, die den einzelnen Methoden übergeben werden können.

2.2.1 RepositoryRegistrationType

Die Anfrageparameter der Repositoryanmeldung. Sollte sich an diesen Parametern nach der Registrierung etwas ändern, ist eine Neuregistrierung des Repositories notwendig. Seine Datenbankeinträge werden dann automatisch überschrieben, ein vorheriges Löschen / Abmelden des Repositories ist nicht nötig.

Title: (string, notwendig) Der Titel dieses Repositories.

Identifier: (string, notwendig) Eindeutiger Bezeichner des Repositories. Soll in Zukunft global für Repository Betreiber generiert und an diese verteilt werden. So können sich nur bekannte / geprüfte Repositories anmelden. Wird anfangs (da wenige, bekannte Repositories anzumelden sind) manuell vergeben.

RepositorySearchSoapAddress: (string, notwendig) Die Webadresse unter der das Repository seinen Search Webservice anbietet.

Description: (string, optional) Volltext Beschreibung des Repositories.

Language: (enum, optional, array) Unterstütze Sprachen dieses Repositories. Auswahl erfolgt aus vorgegebener Liste (GERMAN, ENGLISH, OTHER, UNKNOWN).

Owner: (string, optional) Besitzer / Betreiber des Repositories.

TypesSupported: (enum, notwendig, array) Wählt aus einer vordefinierten Liste von Types (momentan 3D, MUSIC) die vom Repository unterstützten Typen aus.

2.2.2 RepositoryRegistrationQueryEngineType

Der Parametertyp zum Hinzufügen von QueryEngines. Zum Updaten einer QueryEngine muss diese nicht vorher entfernt werden.

RepositoryIdentifier: (string, notwendig) Bezeichner des Repositories, an dem diese Query Engine angemeldet werden soll.

EngineName: (string, notwendig) Name dieser Query Engine.

QueryFormat: (string, notwendig, array) Liste der unterstützen Query Formate. Kann sämtliche MIME Types sowie "'META"' für Metasuche und "'FULLTEXT"' für Volltextsuche enthalten.

2.2.3 RepositoryRemoveQueryEngineType

Der Parametertyp zum Entfernen von Query Engines.

RepositoryIdentifier: (string, notwendig) Bezeichner des Repositories, von dem diese Query Engine gelöscht werden soll.

EngineName: (string, notwendig) Name der zu entfernenden Query Engine.

2.2.4 RepositorySetAvailabilityType

Der Parametertyp, um den Status eines Repositories zu ändern. Wird ein Repository als "'OFFLINE"' markiert, leitet der Core keine Anfragen mehr dahin weiter. Aktualisierung von Metadaten wird weiterhin angenommen.

RepositoryIdentifier: (string, notwendig) Bezeichner des Repositories, dessen Status geändert werden soll.

Availability: (enum, notwendig) Verfügbarkeitsstatus "'ONLINE"' oder "'OFFLINE"'

3 RepositoryMetadataExchange

Der *RepositoryMetadataExchange* Web Service ist Schnittstelle zwischen Repositories (L1) und Core (L2) zum Update / Abbild der Metadaten eines Repositories auf den gemeinsamen Core Metadatensatz. Der *RepositoryMetadataExchange* Web Service wird im Core gehostet, so dass ein Repository per *PUSH* seine Metadaten zum Core schickt und diese somit dort aktualisiert werden. Das *konsumieren* dieses Webservice ist für einen Repository Betreiber also ebenfalls Pflicht, falls er das Core-seitige Durchsuchen seiner Metadaten ermöglichen möchte. EIne Beispielimplementierung dieses WS von Client-Seite aus, ist in Sektion A.6, Seite 18 zu finden.

3.1 Operationen

Dieser Web Service stellt die Operation *doRepositoryMetaDataExchange* bereit.

doRepositoryMetadataExchange: Ein Repository schickt über diese Schnittstelle alle von ihm benötigten Metadaten sowie Identifizierungsinformationen zum Core. Dieser aktualisiert / erweitert die Metadaten in seiner Core Meta DB. Als Antwort liefert diese Methode lediglich *true* oder *false*.

3.2 Parameter

Nachfolgend werden alle Parameter beschrieben, die den einzelnen Methoden übergeben werden können.

3.2.1 RepositoryMetadataExchangeType

Der Typ, der den kompletten Metadatensatz, sowie die Identifikationsdaten für ein Dokument, eines Repository kapselt und so an den Core übermittelt werden kann. Diese Parameter sind auch bei der *ProbadoMetadataSearch* in Form von Key/Value-Pairs zu verwenden, um die Suchanfrage zu spezifizieren.

RepositoryIdentifier: (string, notwendig) Eindeutiger Bezeichner des Repositories, zu dem dieser Block Metadaten gehört.

DocumentIdentifier: (string, notwendig) Eindeutiger Bezeichner für das Dokument (von dem diese Metadaten stammen) innerhalb des Repositories, von dem es kommt. RepositoryIdentifier + DocumentIdentifier bilden den kombinierten Primärschlüssel für die Metadatenbank.

ParentDocumentIdentifier: (string, optional) Der innerhalb des Repositories eindeutige Bezeichner für das logisch-übergeordnete Dokument. So können hierarchische Beziehungen zwischen Dokumenten modelliert werden. Bei 3D kann ein Eintrag in der 3D Datenbank zu einem Modell normalerweise als ein Dokumenteneintrag in die Core Meta DB gespiegelt werden, so das eventuell kein übergeordnetes Dokument vorhanden ist. Bei Musik ist dies durch den werkszentrierten Ansatz i.A. nicht möglich. So gehören zu dem Werk "Beethovens 5te" Unterdokumente (beispielsweise ein gescanntert Notenband, eine CD, eine Midispur, etc.), die sich durch unterschiedliche Metadaten auszeichnen und somit seperat in die Core Meta DB gespiegelt werden. Insbesondere dient der ParentDocumentIdentifier der Gruppierung der Trefferliste.

LinkToPresentation: (string, notwendig) Link (oder Aufrufparameter für ein Applet), dass dem Benutzer das ausgewählte Dokument ("Werk" im Falle von Musik, "Modell" im Falle von 3D) in irgendeiner Form präsentiert. Bei Musik ist dies der DocumentViewer, der verschiedene Abspielmodi des Werkes anbietet, bei 3D kann dies ein einfaches html Dokument sein, dass ein größeres Thumbnail des Modells sowie spezifischere Metadaten und einen Downloadlink des Modells an sich präsentiert. In jedem Fall führt dieser Link zu einer Dokumentenansicht, die vom jeweiligen Repository generiert und gehostet wird. Angezeigt wird dieses jedoch im probado.de Web Rahmen, sollte also vom Design her passend gemacht sein (natürlich nicht erzwingbar / kontrollierbar). Weiterhin muss diese spezielle Dokumentenpräsentation mit dem probado.de Web Rahmen über JavaScript kommunizieren, falls direkt aus ihr heraus weitergesucht werden soll.

PersistentIdentifier: (string, optional) DOI / URN des Dokuments.

Title: (string, notwendig) Der Titel dieses Dokuments.

TypeOfDocument: (enum, notwendig) Typ des Dokuments. Zur Auswahl stehen alle Typen, die vom Probado-System unterstützt werden (Momentan: 3D / MUSIC)

Description: (string, optional) Sonstige Textbeschreibung des Dokuments, die nicht in die anderen Felder passt.

Contributor: (complexType, optional, array)(Besteht aus freiem Textfeld sowie einer Rolle). Eine Person, Firma, Gruppe, etc. die was zu diesem Dokument beigetragen hat. Es können mehrere angegeben werden.

Creator: (string, optional) Die Person, Firma, Gruppe, etc. die hauptverantwortlich für dieses Dokument ist.

Publisher: (string, optional) Die Person, Firma, Gruppe, etc. die das Dokument für Probado zur Verfügung stellt. Textfeld, kann maximal 1 angegeben werden.

RightsHolder: (string, optional) Rechteinhaber des Dokuments. Bspw.: Sony BMG. Textfeld, kann maximal 1 angegeben werden.

Format: (string, notwendig, array) Format (MIMETYPE, Dateiendung, sonstige Konvention) des zum Dokument zugehörigen Datenobjekts (bspw. 3ds)

Subject: (string, optional, array) Kategorie(n) denen das Dokument angehört (bswp. TISCH, NACHTTISCH).

Date: (complexType, optional, array) Verschiedene signifikante Zeitpunkte (samt Beschreibung) die mit dem Dokument verknüpft sind. Bspw.: "11.07.1957, Veröffentlichung auf LP". (enthält Beschreibung, Startzeitpunkt und Endzeitpunkt)

Coverage: (string, optional) Raum und Zeit der von diesem Dokument überspannt wird. Bspw. "später Barock, 1740 bis 1770, Süd-Frankreich".

LinkToThumbnail: (string, optional) Link zu öffentlich zugänglichem Thumbnail. Wird von Core heruntergeladen und selbst gehostet.

BinaryThumbnail: (base64binary, optional) Binären Bilddaten eines Thumbnails. Wird von Core gehostet und öffentlich verfügbar gemacht.

Language: (enum, optional, default=UNKNOWN) Sprache des Dokuments. Aktuell nicht implementiert.

AccessRights: (enum, optional, default=FULL_ACCESS) Feld für zukünftige Rechteverwaltung. Vordefinierte Liste von Rechterollen steht zur Auswahl bereit. Aktuell nicht implementiert.

4 Probado Search

Die Hauptaufgaben des Cores sind zweigeteilt. Zunächst gilt es, Suchanfragen an dafür geeignete Repositories / QE weiterzuleiten. Die Ergebnisse dieser (oder

auch eigener Suchanfragen in der Core Meta DB) gilt es dann als geordnete Ergebnisliste zu aggregieren und zurück zu liefern.

Der *Probado Search* Webservice (`ProbadoSearch.wsdl`) ist die Schnittstelle zwischen dem Frontend (Layer 1) einerseits und dem Core-System (Layer 2) andererseits.

Der Webservice wird vom Core-System angeboten und bietet allen User-Interfaces einen definierten Zugriff auf die Suchfunktionalitäten des Core-Systems. Oberflächen, die mit dem Probado System Nachrichten austauschen möchten, müssen also zwingend diesen Webservice *konsumieren*.

4.1 Operationen

Dieser Webservice stellt zwei verschiedene Operationen zur Verfügung:

doProbadoContentSearch: inhaltsbasierte Suche

doProbadoMetadataSearch: Metadaten-Suche

Beide Methoden liefern eine komplexe XML Nachricht mit der Ergebnisliste der Suchanfrage als Antwort.

4.2 Anfrage-Parameter

4.2.1 ProbadoContentSearchType

Der Typ für den Anfrageparameter der inhaltsbasierten Suche.

QueryData: (base64binary, notwendig) Dieses Element enthält die binären Daten sowie als Attribut den Typ

SortBy: (complexType, optional, array) Mit diesen optionalen Key/Value Elementen kann die gewünschte Sortierreihenfolge der Resultate angegeben werden. Key muss einem CoreMetadaten Eintrag entsprechen, Value ist *ASC* oder *DESC*.

RepositoryId: (string, optional, array) Optional kann angegeben werden, in welchen Repositories zu suchen ist. Ansonsten werden alle angesprochen, die den Anfragetyp unterstützen.

StartIndex (int, optional, default = 0) Anzahl der Treffer die übersprungen werden sollen. Für das *Weiterblättern* von Ergebnislisten.

Count (int, optional, default = 10) die maximale Anzahl an Resultaten. Wird ebenfalls Core-seitig begrenzt.

4.2.2 ProbadoMetadataSearchType

Der Typ für den Anfrageparameter der Metadaten-Suche.

SortBy: (complexType, optional, array) Mit diesen optionalen Key/Value Elementen kann die gewünschte Sortierreihenfolge der Resultate angegeben werden. Key muss einem CoreMetadaten Eintrag entsprechen, Value ist *ASC* oder *DESC*.

RepositoryId: (string, optional, array) Optional kann angegeben werden, in welchen Repositories zu suchen ist. Ansonsten werden alle angesprochen, die den Anfragetyp unterstützen.

StartIndex (int, optional, default = 0) Anzahl der Treffer die übersprungen werden sollen. Für das *Weiterblättern* von Ergebnislisten.

Count (int, optional, default = 10) die maximale Anzahl an Resultaten. Wird ebenfalls Core-seitig begrenzt.

CoreSearch: (bool, optional, default = true) True oder False womit festgelegt wird, ob nur die Core-Metadaten oder auch Repository-spezifische Metadaten durchsucht werden.

Metadata: (complexType, notwendig) Beliebige Metadaten werden als key/value-Paar übergeben. Es muss mindestens ein Metadaten-Element angegeben werden.

Bsp:

```
<Metadata>
<KeyValuePairElement>
<Key>FULLTEXT</Key>
<Value>haus</Value>
</KeyValuePairElement>
</Metadata>
```

4.3 Antwort-Parameter

Sowohl inhaltsbasierte als auch textbasierte Suchanfragen haben folgende Antwortparameter.

4.3.1 ProbadoSearchResultType

TotalResultsCount: (int) Gesamtanzahl der gefundenen Dokumente

Count: (int) Anzahl der zurückgegebenen Dokumente

ResultElement: (complexType, array) Liste mit gefunden Dokumenten. Nachfolgend wird der Inhalt eines solchen Elements aufgeführt

`ResultElement`:

RepositoryIdentifier: (string) Repository aus dem das Dokument stammt

DocumentIdentifier: (string) Bezeichner des Dokuments.

Ranking: (float) Der $[0, 1]$-Ranking-Wert dieses Dokuments. Genaue Verwendung unklar.

Position: (complexType) X/Y Koordinate zum visualisieren des Treffers in einer 2D Darstellung.

Title: (string) Title des Dokuments.

AccessRights: (enum) Für künftige Rechteverwaltung

TypeOfDocument: (enum) Typ des Dokuments

LinkToPresentation: (string) URL auf der das eigentliche DOkument in irgendeiner Art und Weise Präsentiert wird. Sei das direkter Download, Html Preview oder per Plugin. (muss vom Repository bereitgestellt werden!).

LinkToThumbnail: (string) Link zum Thumbnail des Objekts. Wird öffentlich zugänglich auf Core gehostet.

AdditionalBinaryData: (base64binary) Optionales Feld für beliebige Binärdaten. Wird noch nicht verwendet.

5 Repository Search

Der *Repository Search* Webservice (`RepositorySearch.wsdl`) ist die Schnittstelle zwischen dem Core-System (Layer 2) einerseits und einem Repository (Layer 3) andererseits.

Der Web Service wird von jedem Repository angeboten und bietet dem Core-System einen einheitlichen Zugriff auf die Funktionalitäten des Repositories. Dieser Web Service kann als Schnittstelle zu einem Blackbox-Repository dienen. In diesem Fall kümmert sich das Repository selbst um das Weiterleiten von Suchanfragen an seine QE und das Zusammenfassen der zurückgelieferten Ergebnislisten.

Für Repository-Betrieber, die Suche auf ihrem Datenbestand ermöglichen wollen, ist es als zwingend erforderlich einen Webservice gemäß diesen Spezifikationen zu *implmentieren* und zu *hosten*. Bei der Anmeldung an den Core teilen sie diesem die Adresse mit, woraufhin dieser den Webservice *konsumiert*.

5.1 Operationen

Dieser Web Service stellt zwei verschiedene Operationen zur Verfügung:

doRepositoryContentSearch: inhaltsbasierte Suche

doRepositoryMetadataSearch: Metadaten-Suche

Die genaue Implementierung dieser Operationen wird sich natürlich von Repository zu Repository unterscheiden, lediglich auf die Einhaltung des gemeinsamen Ein-/Ausgabeformats ist zu achten.

5.2 Anfrage-Parameter

Die Anfrage-Parameter dieser Methoden unterscheiden sich nur in *einem* Punkt von *ProbadoContentSearchType* bzw. *ProbadoMetadataSearchType*: Der Parameter *RepositoryId* entfällt.

5.3 Antwort-Parameter

Die Antwort-Parameter unterscheiden sich nicht von *ProbadoSearchResultType*.

6 ProbadoFaultType

Der Typ für eine SOAP Fehlermeldung. Dieses Element enthält nur zwei Unterelemente:

Exception: die aufgetretene Exception (stack trace)

ExceptionText: eine optionale textuelle Beschreibung

Dies wird bei sämtlichen auftretenden Fehler verschickt.

A Beispiele

Hier sind Beispiele für die verschiedenen SOAP-Nachrichten angegeben. Der vollständigkeithalber wird das komplette Http POST Request gezeigt, im Regelfall ist aber nur der SOAP Envelope zu generieren.

A.1 Repository Anmeldung

A.1.1 Anfrage

```
POST /RepositoryRegistration HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: "doRepositoryRegistration"
Host: pc1823:9000
Content-Length: 686
Expect: 100-continue
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <RepositoryRegistration xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <Title>repo1</Title>
            <Identifier>MY_REP1</Identifier>
            <RepositorySearchSoapAddress>http://localhost:54320/myRepository</RepositorySearchSoapAddress>
            <Description>test</Description>
            <Owner>max</Owner>
            <TypesSupported>
                <TypesSupportedByProbadoElement>META</TypesSupportedByProbadoElement>
            </TypesSupported>
            <Language>
                <LanguageElement>GERMAN</LanguageElement>
            </Language>
        </RepositoryRegistration>
    </soap:Body>
</soap:Envelope>
```

A.1.2 Antwort

```
HTTP/1.1 200 OK
Content-Length: 421
Content-Type: text/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Wed, 21 Oct 2009 09:59:48 GMT

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <RepositoryRegistrationResponse xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <ResponseValue>true</ResponseValue>
        </RepositoryRegistrationResponse>
```

```
    </soap:Body>
</soap:Envelope>
```

A.2 Query Engine hinzufügen

A.2.1 Anfrage

```
POST /RepositoryRegistration HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: "doRepositoryAddQueryEngine"
Host: pc1823:9000
Content-Length: 668
Expect: 100-continue
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <RepositoryAddQueryEngine xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <RepositoryIdentifier>MY_REP1</RepositoryIdentifier>
            <EngineName>QE1</EngineName>
            <QueryFormat>
                <StringElement>META</StringElement>
                <StringElement>OFF</StringElement>
            </QueryFormat>
        </RepositoryAddQueryEngine>
    </soap:Body>
</soap:Envelope>
```

A.2.2 Antwort

```
HTTP/1.1 200 OK
Content-Length: 465
Content-Type: text/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Wed, 21 Oct 2009 09:59:48 GMT

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <RepositoryAddQueryEngineResponse xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <ResponseValue>true</ResponseValue>
        </RepositoryAddQueryEngineResponse>
    </soap:Body>
</soap:Envelope>
```

A.3 Query Engine entfernen

A.3.1 Anfrage

```
POST /RepositoryRegistration HTTP/1.1
```

```
Content-Type: text/xml; charset=utf-8
SOAPAction: "doRepositoryRemoveQueryEngine"
Host: pc1823:9000
Content-Length: 514
Expect: 100-continue
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <RepositoryRemoveQueryEngine xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <RepositoryIdentifier>MY_REP1</RepositoryIdentifier>
            <EngineName>QE1</EngineName>
        </RepositoryRemoveQueryEngine>
    </soap:Body>
</soap:Envelope>
```

A.3.2 Antwort

```
HTTP/1.1 200 OK
Content-Length: 471
Content-Type: text/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Wed, 21 Oct 2009 09:59:48 GMT

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <RepositoryRemoveQueryEngineResponse xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <ResponseValue>true</ResponseValue>
        </RepositoryRemoveQueryEngineResponse>
    </soap:Body>
</soap:Envelope>
```

A.4 Repository Status ändern

A.4.1 Anfrage

```
POST /RepositoryRegistration HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: "doRepositorySetAvailability"
Host: pc1823:9000
Content-Length: 517
Expect: 100-continue
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
    <soap:Body>
        <RepositorySetAvailability xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <RepositoryIdentifier>MY_REP1</RepositoryIdentifier>
            <Availability>ONLINE</Availability>
        </RepositorySetAvailability>
    </soap:Body>
</soap:Envelope>
```

A.4.2 Antwort

```
HTTP/1.1 200 OK
Content-Length: 467
Content-Type: text/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Wed, 21 Oct 2009 09:59:48 GMT

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <RepositorySetAvailabilityResponse xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <ResponseValue>true</ResponseValue>
        </RepositorySetAvailabilityResponse>
    </soap:Body>
</soap:Envelope>
```

A.5 Metadata hinzufügen

A.5.1 Anfrage

```
POST /RepositoryMetadataExchange HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: "doRepositoryMetadataExchange"
Host: pc1823:9000
Content-Length: 1987
Expect: 100-continue

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
            <RepositoryMetadataExchange xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <RepositoryIdentifier>MY_REP1</RepositoryIdentifier>
            <DocumentIdentifier>MY_DOC1</DocumentIdentifier>
            <ParentDocumentIdentifier/>
            <LinkToPresentation>ftp://test.dyns.org/model.html</LinkToPresentation>
            <PersistentIdentifier>doi</PersistentIdentifier>
            <Title>Riegersburg</Title>
            <Description>Test Description</Description>
            <TypeOfDocument>_3D</TypeOfDocument>
            <Contributor>
                <ContributorElement>
                    <Name>bla1</Name>
```

```
                <Role>rolle</Role>
            </ContributorElement>
        </Contributor>
        <Creator>burgbauer</Creator>
        <AccessRights>FULL</AccessRights>
        <Format>
            <StringElement>META</StringElement>
        </Format>
        <Subject>
            <StringElement>Architektur</StringElement>
            <StringElement>Zeug</StringElement>
        </Subject>
        <Language>UNKNOWN</Language>
        <Publisher>pub</Publisher>
        <Rightsholder>right</Rightsholder>
        <Date>
            <DateTypeElement>
                <Description>zeitspanne</Description>
                <DateBegin>0001-01-01T00:00:00</DateBegin>
                <DateEnd>2009-10-22T15:27:27.137707+02:00</DateEnd>
            </DateTypeElement>
        </Date>
        <Coverage>coverage</Coverage>
        <LinkToThumbnail>www.thumbnails.de/mythumb.jpg</LinkToThumbnail>
    </RepositoryMetadataExchange>
  </soap:Body>
</soap:Envelope>
```


A.5.2 Antwort


```
HTTP/1.1 200 OK
Content-Length: 469
Content-Type: text/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Wed, 21 Oct 2009 11:17:35 GMT

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <RepositoryMetadataExchangeResponse xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <ResponseValue>true</ResponseValue>
        </RepositoryMetadataExchangeResponse>
    </soap:Body>
</soap:Envelope>
```


A.6 RepositoryMetadataExchange Client Code

Hier eine Beispielimplementierung in C# des MetadataExchange WS. Die zugehörigen Metadaten wurden in diesem Beispiel aus einer XML Datei ausgelesen.

```
XDocument musicMeta = XDocument.Load("core_metadata_musik.xml");
var thumbQuery = from attributedItem in musicMeta.Descendants("row")
                 select attributedItem;

foreach (XElement metaElement in thumbQuery)
```

```
{
ChannelFactory<RepositoryMetadataExchangePortType> factory = new ChannelFactory<RepositoryMetadataExchangePortTyp
RepositoryMetadataExchangePortType proxy = factory.CreateChannel();
List<XElement> fields = metaElement.Elements("field").ToList();
ContributorArray contributors = new ContributorArray();
ContributorType contrib = new ContributorType();
contrib.Name = fields[4].Value;
contrib.Role = fields[4].FirstAttribute.Value;
contributors.Add(contrib);
DateTypeArray dates = null;
if (fields.Count > 6)
{
dates = new DateTypeArray();
DateType date1 = new DateType();
date1.DateBegin = new DateTime(Convert.ToInt32(fields[6].Value), 1, 1);
date1.DateEnd = new DateTime(Convert.ToInt32(fields[6].Value), 1, 1);
date1.Description = "";
dates.Add(date1);
}
try
{
/call the WS and all obtained values.
proxy.RepositoryMetadataExchange
(fields[0].Value, fields[1].Value, "", "", "", fields[2].Value, "", TypesSupportedByProbado.MUSIC, contributors,
null, null, Language.UNKNOWN, "", "", dates, "", "");
}
catch (FaultException<ProbadoSoapFaultType> bla)
{
throw bla;
}
//regHttpClient.doRepositoryAddQueryEngine(null);
factory.Close();
 }
```

A.7 Metadata durchsuchen

A.7.1 Anfrage

```
POST /ProbadoSearch HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: "doProbadoMetadataSearch"
Host: pc1823:9000
Content-Length: 963
Expect: 100-continue

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <ProbadoMetadataSearch xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <Metadata>
                <KeyValuePairElement>
                    <Key>TITLE</Key>
                    <Value>riegers</Value>
                </KeyValuePairElement>
```

```
            </Metadata>
            <SortBy>
                <KeyValuePairElement>
                    <Key>CREATOR</Key>
                    <Value>asc</Value>
                </KeyValuePairElement>
            </SortBy>
            <CoreSearch>true</CoreSearch>
            <StartIndex>0</StartIndex>
            <Count>11</Count>
            <SessionId xsi:nil="true"/>
        </ProbadoMetadataSearch>
    </soap:Body>
</soap:Envelope>
```


Bei dieser Anfrage ist *SessionId* nicht mit übertragen worden und wurde entsprechend auf *null* gesetzt. Die Übermittlung von *null*-wertigen Elementen ist stets *optional*.

A.7.2 Antwort


```
HTTP/1.1 200 OK
Content-Length: 1282
Content-Type: text/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Wed, 21 Oct 2009 11:17:35 GMT

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <ProbadoMetadataSearchResponse xmlns="http://www.probado.de/wsdl/ProbadoSchema">
            <TotalResultsCount>1</TotalResultsCount>
            <ResultElements>
                <SearchResultElementArrayElement>
                    <RepositoryIdentifier>MY_REP1</RepositoryIdentifier>
                    <DocumentIdentifier>MY_DOC1</DocumentIdentifier>
                    <Ranking>1</Ranking>
                    <Title>Riegersburg</Title>
                    <AccessRights>FULL</AccessRights>
                    <TypeOfDocument>_3D</TypeOfDocument>
                    <LinkToPresentation>ftp://test.dyns.org/model.jpg</LinkToPresentation>
                    <LinkToThumbnail>www.thumbnails.de/mythumb.jpg</LinkToThumbnail>
                    <AdditionalBinaryData i:nil="true"/>
                </SearchResultElementArrayElement>
            </ResultElements>
            <StartIndex>0</StartIndex>
            <Count>11</Count>
        </ProbadoMetadataSearchResponse>
    </soap:Body>
</soap:Envelope>
```