Technische Universität Dortmund
Fakultät Informatik
Lehrstuhl 5, Prof. Dr. Bernhard Steffen
Wintersemester 2012/13

Aktuelle Themen der Dienstleistungsinformatik

Projekt: Kontakte

Markus Marzotko, Thorben Seeland, Dominic Wirkner
26. Januar 2013

Inhaltsverzeichnis

1 Einleitung

1.1 Das Projekt

Hinführung auf Beschreibung und erste Überlegungen

1.2 Projektbeschreibung

Beschreibung Ahnlich der auf den Folien

1.3 Erste Überlegungen

Aufteilung des Projektes (siehe Grafik). Wahl der Contact-Attribute. Management über SAP-/Google-IDs.

2 Unsere Projektbausteine

Im Folgenden Beschreibung, besondere Überlegungen und Probleme(!)/Entscheidungen der einzelnen Elemente

2.1 SAP-Connector

2.1.1 Aufgabe des SAP Connectors

Die Aufgabe des SAP Connectors besteht darin, die Datenbank im ES Workplace nach einem gegebenen Filter zu durchsuchen. Die gefilterten Datensätze werden als Liste zurückgegeben.

2.1.2 Aufbau und Programmablauf

Alle Operationen dienen dazu, Datensätze aus den Datenbanken des ES Workplace auszulesen. Der ES Workplace stellt hierfür eine Datenbank mit Testdatensätzen bereit, sowie die erforderlichen Webservices, um eine Verbindung

zur Datenbank, die Authentifizierung und die Suche selbst durchzuführen. Ein Webservice beschreibt hier eine Software Anwendung, auf deren Dienste online über eineals XML basierte Schnittstelle zugegriffen werden kann. Die erforderlichen Webservices liegen als XML Dateien vor, werden jedoch zur Verwendung in SIBs als Java Klassen benötigt. Dafür wird Java API for XML Web Services verwendet, welches seit Version 1.6 Bestandteil der Java SE ist. Mittels Jax-WS können in der Konsole durch einen wsimport Befehl alle benötigten Webservices von XML Dateien in Java Klassen umgewandelt werden.

Der SAP Connektor erhält ein Objekt vom Typ `Contact`. Dieses Objekt enthält die vom Benutzer geforderten Filtereinstellungen betreffend Vorname, Nachname, Postleitzahl, Stadt, Straße und Firma. Außerdem wird der Typ nach Kunde, Lieferant und Mitarbeiter unterschieden. Dieses Objekt wird verwendet, um in der Datenbank des ES Workplace, nach dem entsprechenden Typ, Datensätze zu filtern. In Abhängigkeit von der Typbestimmung werden unterschiedliche Webservices ausgeführt:

Lieferant

- *Find Supplier by Name and Address*
- *Read Supplier Basic Data*

Mitarbeiter

- *Find Employee by Elements*
- *Find Employee Address by Employee*

Kunde

- *Find Customer by Elements*

Der Programmablauf soll hier am Beispiel des Kundenwebservices beschrieben werden. Anschließend werden die Besonderheiten bei den Webservices für Mitarbeiter und Lieferant erläutert.

Zunächst müssen die notwendigen Objekte der Klassen des *Webservices Find Customer by Elements* erstellt werden. Mittels einer Klasse die immer mit dem Schlüsselwort *SERVICE* beginnt, kann ein Objekt erzeugt werden auf dem eine getBinding Methode ausgeführt wird. Das erstellte Verbindungsobjekt wird

dann zu einem Objekt vom Typ Binding Provider gecastet, auf welchem dann mittels `getRequestContext()` Methoden Passwort und Username gesetzt werden können. Für die Authentifizierung ist normalerweise auch noch eine Webadresse anzugeben, allerdings ist bei den verwendeten Webservices die Adresse der Testdatenbank bereits enthalten. Das Verbindungsobjekt verfügt über eine einzige Methode, die ein Objekt mit Suchkriterien übergeben bekommt und eine Liste von Suchergebnissen zurückliefert. Die Suchkriterien können mit bereits integrierten set Methoden gesetzt werden, die Suchergebnisse über get Methoden ausgelesen werden. Anschließend werden die Suchergebnisse als Liste von Kontaktobjekten zurückgegeben.

Die Webservices *Find Supplier by Name and Address* und *Find Employee by Elements* funktionieren ähnlich, liefern allerdings nur einle Liste mit Namen und SAP-IDs. Die SAP-IDs sind eindeutige Nummern, die jedem Datensatz zugeordnet werden. Den Webservices *Read Supplier Basic Data* und *Find Employee Address by Employee* ist es möglich die fehlenden Adressdaten mit den SAP-IDs auszulesen. Dies muss jedoch für jede SAP-ID separat geschehen, also muss für jede SAP-ID von Lieferant und Mitarbeiter der entsprechende Webservice einzeln aufgerufen werden, die Übergabe einer Liste von IDs ist nicht vorgesehen. Danach können die Suchergebnisse auch hier als Liste von Kontaktobjekten zurückgegeben werden.

2.1.3 Probleme

Die Verwendung der Webservices des ES Workplace bringt einige Herausforderungen mit sich, was Programmierung und Anwendung angeht. Die Programmierung wird durch die hohe Anzahl unterschiedlicher Klassen in den Webservices erschwert. Da für jeden Webservice eigene Objekte erstellt werden müssen, die in den anderen Webservices, selbst bei gleichem Inhalt wie PLZ oder Stadtname, nicht wiederverwendet werden können, ist eine prozeduraler Programmieransatz die beste Lösung. Eine objektorientierte Lösung ist mittels vieler Interfaces, sehr guter Planung und hohem Entwicklungsaufwand zwar auch realisierbar, ob die leicht verbesserte Wiederverwendbarkeit den Aufwand allerdings rechtfertigt ist fraglich. Zumal lediglich der Verbindungsaufbau in allen WSDLs als ähnlich zu betrachten ist, selbst get und set

Methoden sind völlig unterschiedlich. Im Beispielcode 1 müssen im Webservice *Find Employee by Elements* erst drei Objekte der inneren Klassen erstellt werden, um im innersten Objekt einen String zuzuweisen und dann die Objekte den Objekten der äußeren Klassen zuzuweisen.

```
CustomerSelectionByNameAndAddress kundeFilter = new
   CustomerSelectionByNameAndAddress();
AddressInformation add1 = new AddressInformation();
Address add2            = new Address();
PhysicalAddress add3    = new PhysicalAddress();

add3.setCityName("Hamburg");
add2.setPhysicalAddress(add3);
add1.setAddress(add2);

kundeFilter.setAddressInformation(add1);
```

Listing 1: Beispiel für eine einfache Zuweisung

Ein weiteres Problem stellen die separaten Einzelaufrufe der Webservices für jede ID dar1. Sollten dem Benutzer keine vollständigen Informationen vorliegen, könnte ggf. mehr als ein Suchergebnis zurückgegeben werden. Jeder Webserviceaufruf bedingt zusätzliche Authentifizierungen und Datenbankzugriffe, die in keinem Verhältnis zum verwendeten Ergebnis stehen.

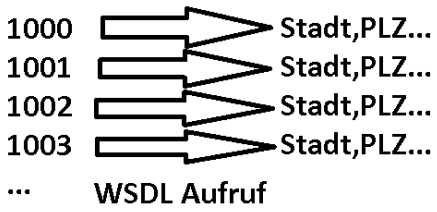

Abbildung 1: Bild1

2.1.4 Ausblick

Im Folgenden soll eine Lösung für die Mehrfachaufrufe der Webservices beschrieben werden.

Der erste Webserviceaufruf gibt uns, neben der SAP ID, auch den Namen zurück. Damit hat der Benutzer ein Identifikationsmerkmal, welches eine Vorentscheidung, betreffend die zu ladenden Adressdaten, ermöglicht. Wir müssen dem Benutzer jetzt also ermöglichen, nur für von ihm ausgewählte Namen die Adressdaten zu laden. Das Graphical User Interface soll nach dem erstem Webserviceaufruf eine Liste aller Namen anzeigen, aus welcher der Benutzer einen auswählt. Beim Klick auf einen Namen liefert der Webservice, mittels der zugehörigen ID, die Adressdaten zurück 2. Dadurch werden die Authentifizierungen und Datenbankzugriffe reduziert.

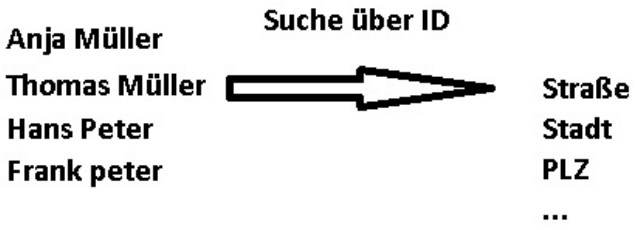

Abbildung 2:

2.2 Google-Connector

2.2.1 Die Bibliothek gdata

Die Google-Bibliothek *gdata* ist eine frei verfügbare Bibliothek zum erstellen von Clientapplications für die Services der Google-Cloud. *gdata* kapselt

die Webservices komplett in Java-Klassen, so dass ein importieren (z. B. mit *wsimport*) nicht mehr notwendig ist.

In der Beschreibung der *gdata*-Bibliothek wird beschrieben, wie man aus der zip-Datei ein JAR compilieren kann, da dies bei mir in mehreren Versuchen nicht geklappt hat, haben wir die pragmatische Lösung gewählt und die in der zip-Datei enthaltenen JARs einzeln in das Projekt eingefügt[6].

2.2.2 Authentifizieren und Verbinden mit gdata

Google bietet zwei Authentifizierungsverfahren an

1. *OAuth*
2. Username und Passwort

OAuth ist ein Service, der bei erfolgreicher Anmeldung ein Token erstellt, mit dem der Client von Google bereitgestellte Services aufrufen und sich authentifizieren kann. So muss der Client die Anmelde-Daten des Nutzers nicht speichern, sondern nur den Token. In Abbildung 3 wird ein Beispiel für die Nutzung von *OAuth* dargestellt.

Da wir unseren eigenen Account nutzen und die Daten nicht sicherheitskritisch sind, haben wir die zweite Variante gewählt und authentifizieren uns bei jedem Service-Aufruf mit Username und Passwort.

Die Authentifizierung wird für ein `ContacsService`-Objekt, wie in Listing 2 abgebildet, einmal durchgeführt, danach wird sie vom Framework automatisch durchgeführt.
In der Abbildung wird ein `ContactsService`-Objekt erstellt. Der `String`-Parameter `servicename` dient hier als Identifikator für den Nutzer.
Nachdem das `ContactsService`-Objekt erstellt ist, wird der Service mit Nutzername und Passwort angemeldet.

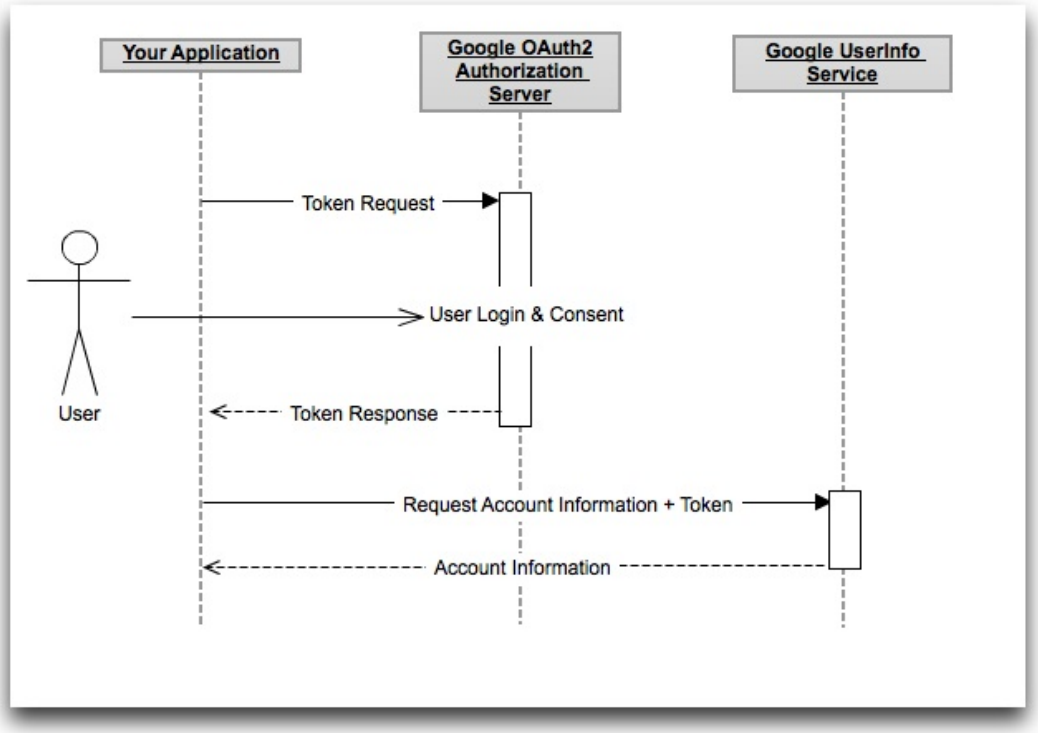

Abbildung 3: Nutzungsbeispiel für *OAuth*[5]

```
ContactsService myService;
myService = new ContactsService(servicename);
try {
        myService.setUserCredentials(username, password);
} catch (AuthenticationException e) {
        e.printStackTrace();
}
```

Listing 2: Beispiel für die Authentifizierung ohne *OAuth*

2.2.3 Kontakte suchen

Die *gdata*-Bibliothek bietet die Möglichkeit, Kontakte wie in Listing 3 mit Angabe eines `Query`-Objekts herunterzuladen. Im Codebeispiel wird das `Query`-Objekt `myQuery` für die hinter der URL `feedURL` liegenden Kontakte initialisiert, danach wird es durch die `"group"`-Option angewiesen nur Kontakte aus der Gruppe mit dem `String`-Identifikator `groupId` herunterzuladen. Nachdem `myQuery` konfiguriert ist, wird mit dem authentifizierten `ContactsService`

myService der Service-Aufruf per query()-Befehl ausgeführt. Die Klasse Query kann allerdings nur zwischen Gruppen unterscheiden, jedoch nicht nach anderen Kriterien wie z. B. dem Vornamen oder dem Nachnamen eines Kontakts filtern.

```
URL feedUrl = new URL(contactsURL);
Query myQuery = new Query(feedUrl);
ContactFeed resultFeed = null;
// Gruppe
String groupId = null;
// Parameter Contact filter
switch (filter.getType()) {
case CUSTOMER:
        groupId = customerGroupURL;
        break;
case SUPPLIER:
        groupId = supplierGroupURL;
        break;
case EMPLOYEE:
        groupId = employeeGroupURL;
        break;
default:
        break;
}
myQuery.setStringCustomParameter("group", groupId);
// submit request
resultFeed = myService.query(myQuery, ContactFeed.class);
```

Listing 3: Kontaktsuche per Query

Das Suchen von Kontakten geschieht in unserem Projekt durch das Herunterladen aller Kontakte einer Gruppe und anschließendem Aussortieren „von Hand“.

2.2.4 Kontakte einfügen

Kontakte werden in Google Contacts eingefügt, indem ein Objekt der Klasse ContactEntry erstellt, mit den gewünschten Daten befüllt und durch ein ContactsService-Objekt versendet wird.
In dem Beispielcode der Listings 4-8 wird vorgeführt, wie aus dem Datencon-

tainer `contactInfo`, der alle relevanten Kontaktinformationen enthält, ein `ContactEntry`-Objekt erstellt wird.

Die Klasse `ContactEntry` stellt drei Möglichkeiten bereit, Daten einzutragen. Die Möglichkeit, die in Listing 4 zuerst gezeigt wird, ist ein Feld vom Typ `String` direkt mit einem Wert zu füllen, hier wird exemplarisch der Titel des Eintrags gesetzt. Dies ist jedoch nur für bestimmte von Google vorgesehene Felder so.

```
// Create the entry to insert
ContactEntry contact = new ContactEntry();
contact.setTitle(new
   PlainTextConstruct(contactInfo.getFirstname()
                + contactInfo.getLastname()));
```

Listing 4: Kontakt-Objekt (`ContactEntry`) erstellen

Die zweite Möglichkeit ist eine von Google vorgesehene Struktur einzutragen, dies betrifft den Namen, Telefonnummer, E-mail und die Adresse. In Listing 5 wird der Name des Kontakts erstellt. Der Inhalt von `Name` wird in der Weboberfläche von Google Contacts als Titel verwendet, deshalb fordert der Service, dass `Name` einen `String`-Inhalt hat, der nicht leer ist. Hier wurden `FamilyName` und `GivenName` gesetzt.

```
// Name
Name name = new Name();
name.setFamilyName(new
   FamilyName(contactInfo.getLastname()));
name.setGivenName(new
   GivenName(contactInfo.getFirstname()));
contact.setName(name);
```

Listing 5: Namen in ein Kontakt-Objekt (`ContactEntry`) einfügen

Die dritte Möglichkeit sind benutzerdefinierte Inhalte, wie beispielsweise die Firma oder die SAP-ID des Kontaktes. Benutzerdefinierte Inhalte werden von der Klasse `ExtendedProperty` dargestellt, in der der Name und der Wert des Eintrags gesetzt werden können. Im Listing 6 wird beispielhaft die Firma `"Company"` des Kontakts gesetzt wird.

```
// Firma
if (!contactInfo.getCompany().isEmpty()) {
        ExtendedProperty company = new ExtendedProperty();
        company.setName("Company");
        company.setValue(contactInfo.getCompany());
        contact.addExtendedProperty(company);
}
```

Listing 6: Benutzerdefinierte Einträge zu einem Kontakt-Objekt (`ContactEntry`) hinzufügen

Um die in der Aufgabenstellung geforderte Unterteilung der Kontakte in Lieferanten, Kunden und Angestellte zu realisieren, haben wir die von Google bereitgestellten Gruppen genutzt. Um den Kontakt, wie in Listing 7 dargestellt, einer Gruppe hinzuzufügen muss die URL der Gruppe als Identifikator der Gruppe angegeben werden. Bei dem Aufruf `new GroupMembershipInfo(false, groupURL)` steht das `false` dafür, dass die Gruppe, der der Kontakt angehört nicht gelöscht wurde.

```
// Gruppe setzen
String groupURL = null;
groupURL = customerGroupURL;
contact.addGroupMembershipInfo(new
   GroupMembershipInfo(false, groupURL));
```

Listing 7: Den Kontakt einer Gruppe hinzufügen

Ist das `ContactEntry`-Objekt befüllt, muss es unter Angabe der URL, hinter der sich die Kontakt-Datenbank befindet, an den Webservice weitergegeben werden, wie in Listing 8 abgebildet.

```
// Kontakt senden
URL postUrl = new URL(contactsURL);
return myService.insert(postUrl, contact);
```

Listing 8: Das Kontakt-Objekt (`ContactEntry`) an den `ContactsService myService` übergeben

2.3 SIB-Programmierung

Generelles zum Aufbau von SIBs. Besonderheiten

2.4 Das Modell im jABC

2 GUI-Elemente. Gemeinsamkeiten/Konzeption. Validierung in EDIT. Details in CHOOSE.

2.5 Zusammenfassung

3 Zusammenfassung / Eigene Meinung / Weiterführendes

...

4 Literaturverweise

Literatur

[1] Wasserman, Stanley und Katherine Faust; *Social Network Analysis: Methods and Applications*; Cambridge University Press, New York; 1995

[2] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest und Clifford Stein; *Introduction to Algorithms, Second Edition*; The MIT Press and McGraw-Hill Book Company; 2001

[3] Jacob, Riko, Dirk Koschtzki, Katharina Anna Lehmann, Leon Peeters und Dagmar Tenfelde-Podehl; *Algorithms for Centrality Indices*;
In: Brandes, Ulrik und Thomas Erlebach

[4] Brandes, Ulrik und Thomas Erlebach; *Network Analysis: Methodological Foundations*; Band 3418 der Reihe *Lecture Notes in Computer Science*; Springer; 2005

[5] Google Developers; *Using OAuth 2.0 to Access Google APIs*; `https://developers.google.com/accounts/docs/OAuth2`; 26. Januar 2013

[6] Google Code; *gdata Java Client*; `http://code.google.com/p/gdata-java-client/downloads/list`; 26. Januar 2013