

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
29. Januar 2013

Inhaltsverzeichnis

1	Einleitung	3
1.1	Projektbeschreibung	3
1.2	Erste Überlegungen	4
1.2.1	Projektstruktur	5
1.2.2	Das Contact-Objekt als Schnittstelle	6
1.2.3	Eingesetzte Technologien	6
2	Unsere Projektbausteine	7
2.1	SAP-Connector	8
2.1.1	Aufgabe des SAP Connectors	8
2.1.2	Aufbau und Programmablauf	8
2.1.3	Probleme	10
2.1.4	Ausblick	11
2.2	Google-Connector	12
2.2.1	Die Bibliothek <i>gdata</i>	12
2.2.2	Authentifizieren und Verbinden mit <i>gdata</i>	12
2.2.3	Kontakte suchen	14
2.2.4	Kontakte einfügen	15
2.3	SIB-Programmierung	17
2.3.1	Allgemeines zur SIB-Programmierung	18
2.3.2	Spezifikation notwendiger SIBs	19
2.3.3	Besonderheit der GUI-SIBs	23
2.3.4	Die JAVA-Laufzeitumgebung in jABC	24
3	Der fertige Prozess im jABC	25
4	Beurteilung des Projektes	26
5	Literaturverweise	27

1 Einleitung

Als Abschlussübung der Vorlesung *Aktuelle Themen der Dienstleistungs-informatik* im Wintersemester 2012/13 an der TU-Dortmund sollten die teilnehmenden Studenten ein Projekt zum Thema Webservices durchführen. Dieser Bericht erläutert nun im Detail, wie das Projekt *Kontakte* durchgeführt wurde.

In Kapitel 1.1 wird zunächst die genaue Aufgabenstellung erläutert, welche das eigentliche Projektziel darstellt. In diesem Zusammenhang werden auch erste Überlegungen zur Projektplanung und -struktur aufgeführt.

Daraufhin werden in Kapitel 2 die konkreten Projektbausteine behandelt. Zunächst werden die beiden Schnittstellen zu SAP und Google, welche auf Webservices basieren, im Detail erläutert, bevor es im Anschluss mit dem Thema der SIB-Programmierung weitergeht. Es wird erläutert, wie die SIB-Programmierung im Detail funktioniert und inwiefern die oben beschriebenen Schnittstellen in diesem Bereich ihre Anwendung finden.

Kapitel 3 beschäftigt sich mit dem konkreten Projektergebnis: dem fertigen jABC-Modell. Es wird der durch das Modell beschriebene Prozess im Detail erläutert und einige Fragen des Modelldesigns behandelt.

Abschließend folgt in Kapitel 4 die konkrete Betrachtung der Projektergebnisse, um den Erfolg des Projektes zu beurteilen.

1.1 Projektbeschreibung

Als Grundlage für die Aufgabenstellung diente ein fiktives Szenario in Form einer Umstellung der informationstechnischen Infrastruktur eines Unternehmens. Konkretes Thema für dieses Projekt war die Migration von kontaktbezogenen Stammdaten aus der Datenbank von SAP in das System von Google. Für die Migration mussten drei Gruppen von Kontakten behandelt werden: Kunden, Lieferanten und Mitarbeiter. Die Migration selbst sollte dabei nicht der trivialen Strategie folgen, alle vorhandenen Kontakte in einem Prozess komplett zu kopieren. Aufgrund der Existenz von sogenannten „Karteileichen“ in der Datenbank von SAP, wurde eine Strategie vorgegeben, welche eine Mi-

gration nur bei Bedarf vorsieht. Dadurch wird garantiert, dass das System von Google nur Kontaktdaten enthält, welche aktiv von dem Unternehmen genutzt werden.

Die Migrationsstrategie behandelt demnach nur einen einzelnen Kontakt und gliedert sich in drei Teile. Zunächst muss im System von Google überprüft werden, ob der gewünschte Kontakt schon migriert worden ist. Ist dies der Fall, so findet keine erneute Migration statt und der Prozess ist beendet.

Falls der gesuchte Kontakt nicht im System von Google existiert, wird versucht, diesen in der Datenbank von SAP zu finden und im Anschluss nach Google zu übertragen. Da es bei einer Suche nach einem Kontakt, abhängig von den Kriterien, sowohl bei Google als auch bei SAP dazu kommen kann, dass mehrere Ergebnisse zurückgeliefert werden, muss es dem Nutzer möglich sein, den gewünschten Kontakt aus dieser Ergebnisliste manuell auszuwählen. In diesem Fall endet der Prozess mit einer erfolgreichen Migration von SAP nach Google.

Zudem muss die Tatsache abgedeckt sein, dass der gesuchte Kontakt ein gänzlich neuer ist, welcher bisher noch in keiner Datenbank auftaucht. Dann soll der Kontakt nach manueller Dateneingabe bei Google angelegt werden.

Die Implementierung des Projektszenarios soll dabei als Prozessmodell in der Software jABC erfolgen. jABC ist ein Akronym und steht für „Java Application Building Center“. Mit Hilfe dieser Software ist es möglich Prozesse als Graphen zu modellieren. Ein Knoten dieses Graphen wird in diesem Kontext auch SIB genannt, welches ebenfalls ein Akronym ist und für „Service Independent Building Blocks“ steht.

1.2 Erste Überlegungen

Damit eine effiziente Projektplanung gewährleistet werden kann, muss zunächst die Aufgabenstellung analysiert werden. Das Ziel dieser Analyse ist es zunächst, Bereiche zu identifizieren, welche unabhängig bearbeitet werden können. Dies unterstützt die Aufgabenverteilung an die einzelnen Projektmitglieder und legt deren Verantwortungsbereiche fest.

Da diese Teilbereiche im Projektzusammenhang jedoch an einigen Stellen miteinander interagieren sollen, müssen zudem geeignete Schnittstellen an den Berührungspunkten definiert werden.

Zudem gilt es auch, sich am Anfang der Planung über einzusetzende Technologien zu einigen, welche die Entwicklung unterstützen.

1.2.1 Projektstruktur

Aufgrund der Aufgabenstellung lassen sich die beiden Schnittstellen zu den Datenbanksystemen leicht als zwei unabhängige Bereiche identifizieren. Ein weiterer Bereich ergibt sich durch die Tatsache, dass Nutzereingaben behandelt werden müssen. Die Erstellung von GUI-Elementen bildet also den dritten Bereich des Projektes.

Diese soeben definierten Unterstrukturen der Software finden ihre Verwendung in der SIB-Programmierung, dessen Ergebnis das Produkt des Projektes darstellt: das jABC-Modell. Somit wird der letzte eigenständige Bereich durch die Generierung dieser SIBs klassifiziert. Die Abbildung 1 zeigt eine sich daraus ergebene Projektstruktur.

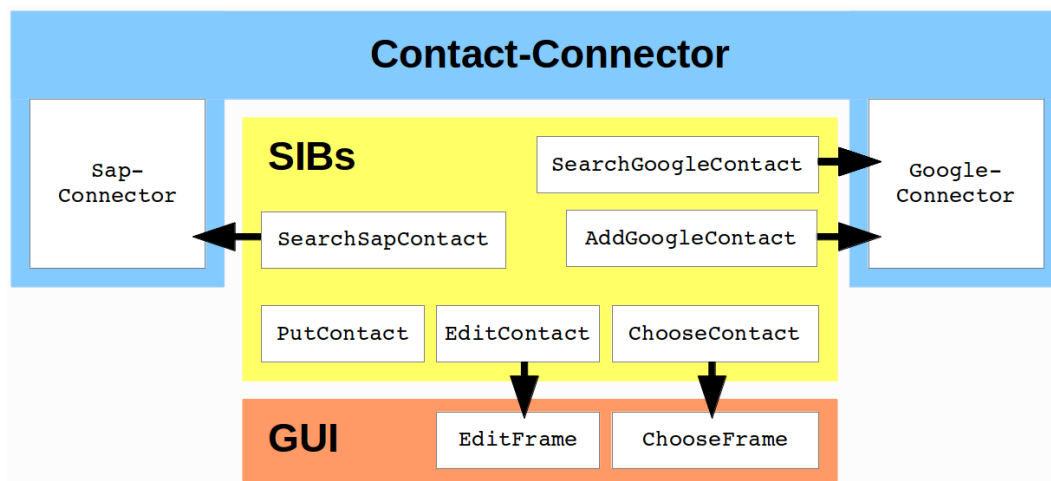


Abbildung 1: Projektaufbau

1.2.2 Das Contact-Objekt als Schnittstelle

Die Verwendung der Unterstrukturen in der SIB-Programmierung macht es notwendig, eine geeignete Kommunikationsschnittstelle zu definieren. Die Aufgabenstellung macht deutlich, dass der primäre Gegenstand des Prozesses ein Contact-Objekt ist. Kontaktdaten werden eingegeben und an die Programmteile weitergegeben, welche die Kommunikation mit den externen Datenbanken realisieren. Daraufhin wird eine Liste von Contact-Objekten zurückgeliefert und ein ausgewählter Kontakt wird dann bei Google gespeichert.

Ein einziges Contact-Objekt kann also als Schnittstelle genutzt werden, wenn es hinreichend klar definiert ist. Das heißt, dass eine Abbildung gefunden werden kann, welche die Attribute des Contact-Objektes mit den speziellen Strukturen der externen Datenbanken verknüpft. Aus diesem Grund wurden für das Projekt Attribute gewählt, welche in beiden Datenbanken verwendet werden. Insbesondere musste auch darauf geachtet werden, dass diese Attribute in allen drei Gruppen (Kunde, Lieferant, Mitarbeiter) vorhanden sind. Aus diesem Überlegungen wurde das Contact-Objekt mit folgenden Attributen definiert:

- Vorname, Nachname, Firma
- Adresse (Straße, PLZ, Ort)
- Telefonnummer, Emailadresse

Des weiteren enthält das Kontaktobjekt die eindeutigen Identifikationskennungen des SAP- und Google-Systems. Zum einen kann dadurch die Herkunft eines Kontakts verfolgt werden und zum anderen bietet es Möglichkeiten zur Implementierung einer komplexeren Migrationsstrategie. Es kann dann zum Beispiel automatisch geprüft werden, ob ein Kontakt bereits migriert wurde oder ob dieser Kontakt überhaupt aus dem SAP-System stammt.

1.2.3 Eingesetzte Technologien

Es wurden einige Entscheidungen über zu verwendende Tools getroffen, um die Projektarbeit und die Implementierung zu unterstützen.

Zum einen wurde das Versionskontrollsystem *Git* verwendet. Neben einer zentralen Speicherung aller Projektdaten unterstützt *Git* die kooperative Implementierung des Programmcodes. Änderungen an verschiedenen Teilen der Software werden automatisch zu einer neuen Version zusammengefasst.

Um die einzelnen Programmbausteine ausreichend zu testen, um deren Funktion zu verifizieren, wurde zudem das Test-Framework *JUnit* verwendet.

Dadurch ist es beispielsweise möglich, die Suchfunktion der Konnektoren ausgiebig zu testen, bevor diese in der SIB-Programmierung ihre Verwendung finden.

Ein weiteres Tool, welches zur Unterstützung für den Softwarelebenszyklus verwendet wurde, ist *Apache Maven*. Es bietet den Vorteil, dass im Anschluss an die Kompilation automatisch *JUnit*-Tests ausgeführt werden, bevor die Software zu einem großen Paket geschnürt wird. *Apache Maven* trägt außerdem dafür Sorge, dass alle Abhängigkeiten, welche für die Ausführung der Software notwendig sind, mit in das erzeugte Programmpaket gepackt werden. Dies erleichtert die anschließende Verwendung im jABC, da nur ein einziges Paket eingebunden werden muss.

Dazu muss jedoch gesagt werden, dass das Zusammenschnüren von einem großen Paket im allgemeinen nicht empfohlen wird. Falls eine einzelne Komponente des Pakets aktualisiert werden soll, muss das ganze Paket neu erstellt werden.

2 Unsere Projektbausteine

Dieses Kapitel beschreibt die Projektbausteine und weist auf die entdeckten Probleme und getroffenen Entscheidungen hin.

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 eine auf 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* (*JAX-WS*) 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 Connector 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 `BindingProvider` Methode ausgeführt wird. Das erstellte Verbindungsobjekt wird dann zu einem Objekt vom Typ `BindingProvider` 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 eine 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 `Contact`-Objekten 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 Listing 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 dar, siehe Abbildung 2. 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.

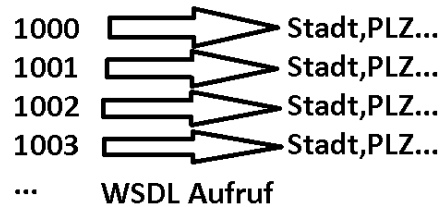


Abbildung 2: Mehrfacher WSDL Aufruf

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, siehe Abbildung 3. Dadurch werden die Authentifizierungen und Datenbankzugriffe reduziert.

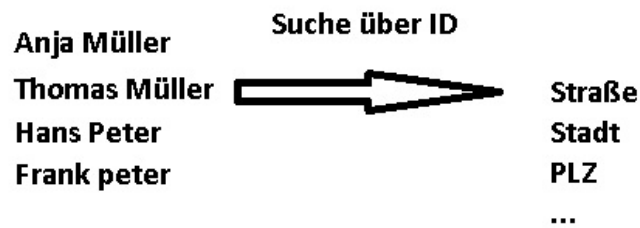


Abbildung 3: Einfacher WSDL Aufruf

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[2].

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 4 wird ein Beispiel für die Nutzung von *OAuth* dargestellt.

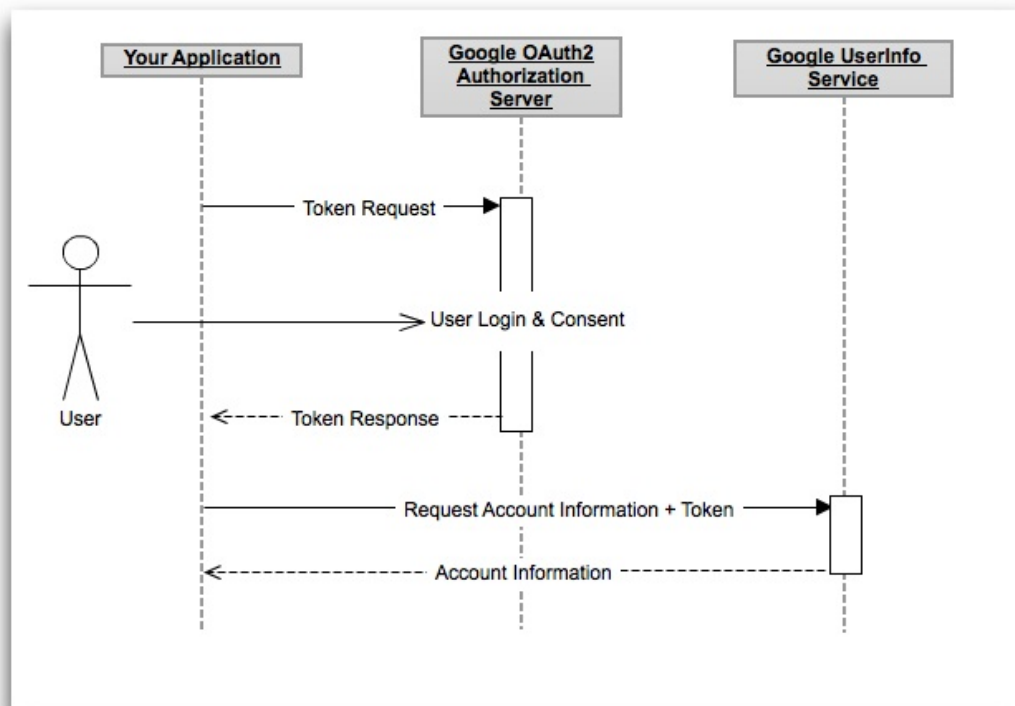


Abbildung 4: Nutzungsbeispiel für *OAuth*[1]

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 `ContactsService`-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.

```
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.

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

```
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

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 Datencontainer `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.

Die zweite Möglichkeit ist eine von Google vorgesehene Struktur einzutragen,

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

Listing 4: Kontakt-Objekt (ContactEntry) erstellen

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

In diesem Kapitel wird zunächst auf die Implementierung von SIBs im allgemeinen eingegangen. Darauf folgt die genaue Beschreibung der für das Projekt erstellten SIBs, sowie die Darstellung zweier Besonderheiten im diesem Entwicklungsprozess.

2.3.1 Allgemeines zur SIB-Programmierung

Im jABC können SIBs über eine Baumstruktur ausgewählt und dann bequem in das Prozessmodell eingebunden werden. Um die verwendeten SIBs miteinander zu verbinden, besitzt jedes SIB fest definierte ausgehende Kanten, im jABC-Kontext auch Branches genannt. Zudem ist es üblich, dass ein SIB definierte Parameter benötigt, um dessen Ausführung zu steuern. Je nach Zweck des einzelnen SIB unterscheiden sich daher dessen Branches und Parameter. Im folgenden wird nun erläutert, wie ein solches SIB erstellt wird.

Da jABC eine Java-Anwendung ist und die Ausführung des Prozessmodells, und dadurch auch die Ausführung der einzelnen SIBs an sich, innerhalb von jABC stattfindet, werden die einzelnen SIBs ebenfalls in Java implementiert. Ein SIB wird dabei durch eine Java-Klasse repräsentiert, welche mit der Annotation `@SIBClass` versehen ist. Durch diese Annotation wird die Java-Klasse von jABC als SIB erkannt.

Damit das SIB innerhalb einer Modellausführung verwendet werden kann, müssen jedoch zunächst noch Branches und Parameter definiert werden. Mögliche Branches werden dabei über eine öffentliche Klassenvariable namens `BRANCHES` definiert, welche vom Typ `String[]`-Array ist. Auch SIB-Parameter sind öffentliche Klassenvariablen. Der Name der Variablen ist beliebig und es kann aus den üblichen Standard-Typen (z.B. `boolean`, `int`, `String`, ...) ausgewählt werden. Für die Notwendigkeit der Anwendung von komplizierteren Parametern, wie ganzen Klassen, stellt jABC den Variablentyp `ContextKey` zur Verfügung. Mit diesem ist es möglich Objekte innerhalb des Ausführungskontextes von jABC zu referenzieren. Diese können dann lesend wie schreibend verwendet werden. In jABC ist der Ausführungskontext über eine Map realisiert. Daher auch der Name `ContextKey` des Variablentyps, denn bei der Deklaration einer Variable von diesem Typ ist lediglich der zugehörige Schlüssel aus der Map anzugeben. Die Klassenvariable kann dann dazu verwendet werden, auf die Daten im Ausführungskontext zuzugreifen.

Jedoch bedarf es noch einer weiteren Ergänzung, um ein SIB vollständig zu implementieren. Es muss noch definiert werden, was dieses SIB eigentlich tun soll. Hierfür muss die Java-Klasse des SIBs das Interface `Executable` imple-

mentieren, welches verlangt, dass die Klasse die Methode `trace()` definiert. Diese Methode `trace()` wird innerhalb von `jABC` in dem Moment aufgerufen, wenn die Ausführung des SIBs beginnen soll. Als Parameter wird eine Variable vom Typ `ExecutionEnvironment` übergeben, welche für den Zugriff auf den Ausführungskontext erforderlich ist. Zuletzt muss `jABC` noch mitgeteilt werden, welcher Branch im Anschluss an die Ausführung gewählt werden soll. Die Methode `trace()` besitzt hierzu den Rückgabewert `String`. Überlicherweise wird der Rückgabewert aus dem wie oben definierten `String[]`-Array `BRANCHES` ausgewählt.

Das Listing 9 zeigt exemplarisch den Kopf einer typischen Implementierung einer SIB-Klasse.

```
@SIBClass("My-SIB")
public class MySib implements Executable {

    // Branches
    public final String[] BRANCHES = {"default", "error"};

    // Parameter
    public \lstinline{ContextKey} someKey = new
        ContextKey("someKey");
    public String title = "Nice Title";

    @Override
    public String trace(ExecutionEnvironment env) {
        ...
        return "default";
    }
    ...
}
```

Listing 9: Implementierung der SIB-Klasse

2.3.2 Spezifikation notwendiger SIBs

Wie bereits aus der Projektstruktur in Abbildung 1 ersichtlich, sind verschiedene Arten von SIBs für ein geeignetes Modell notwendig. Zum einen werden SIBs benötigt, welche die Funktionen der Konnektoren benutzen. Zum anderen

müssen SIBs implementiert werden, welche über GUI-Elemente die Eingaben des Nutzers abfragen. Daraus ergibt sich die Notwendigkeit folgender SIBs.

Ein zentrales Element des Prozesses ist die Suche nach Kontakten anhand gegebener Kriterien. Daher werden für die beiden Suchfunktionen (bei SAP und Google) passend zwei SIBs implementiert. Die beiden SIBs unterscheiden sich jedoch nur in der Form, dass sie bei der Ausführung die jeweils passende Suchmethode der externen Datenbank verwenden. Äußerlich unterscheiden sich diese, bis auf den Namen, daher nicht. Als Eingabe erwarten beide Suchfunktionen einen Parameter vom Typ des in der Einleitung beschriebenen `Contact`-Objektes. Zurück liefern beide Funktionen eine Liste von eben genannten `Contact`-Objekten. Parameter und Rückgabewert sind als SIB-Parameter vom Typ `ContextKey` implementiert, wodurch eine weitere Verwendung möglich wird. Definierte Branches der beiden SIBs sind *found* (für den Fall dass die zurückgelieferte Liste nicht leer ist), *not found* (im Falle einer leeren Rückgabeliste) und *error* (wird gewählt, wenn die Ausführung der Suchfunktion eine `Exception` wirft).

Ein weiteres Element einer Datenmigration ist das Hinzufügen der Objekte auf dem Zielsystem. Zu diesem Zweck muss ein SIB erzeugt werden, welches die passende Funktion nutzt, um Kontakte dem Google-System hinzuzufügen. Der Parameter des SIB ist in diesem Fall vom Typ `ContextKey` und verweist auf ein `Contact`-Objekt, dessen Variablen mit passenden Werten belegt sind. Ein spezieller Rückgabewert ist in diesem Fall unnötig, denn Erfolg oder Misserfolg der Funktionsausführung wird über den gewählten Branch kommuniziert. Zu diesem Zweck besitzt das SIB die zwei Branches *default* (falls kein Fehler auftritt) und *error* (falls `Exception` geworfen wird, s.o.).

Damit ist die Kommunikation zu den externen Datenbanken ausreichen spezifiziert und es folgt die Definition der SIBs, welche für die Nutzereingaben verantwortlich sind.

Damit der Nutzer eine Suchanfrage definieren kann, muss jener die Daten eingeben können. Hierzu wird ein SIB generiert, welches ein Fenster (Swing-Frame) öffnet und anschließend auf die Eingabe des Nutzers wartet. Die Eingabefelder können mit den Werten eines `Contact`-Objektes vorbelegt werden. Wenn

die Eingabe beendet ist, speichert das SIB die Daten in den Ausführungs-kontext. Zu diesem Zweck wurde ein passender SIB-Parameter vom Typ `ContextKey` definiert. Dieser dient sowohl für die Vorbelegung, als auch zur Speicherung der Eingaben. Um die Wiederverwendbarkeit zu ermöglichen, wurde das SIB um weitere Parameter erweitert. Zum einen besitzt das `Contact`-Objekt eine Funktion namens `validate()`, mit Hilfe derer sich die Eingaben des Nutzers kontrollieren lassen. Diese kann mittels eines Parameters vom Typ **boolean** an- bzw. ausgeschaltet werden. Auf diese Weise ist es möglich das SIB sowohl für die Suchanfrage (keine Validierung notwendig) als auch zur Dateneingabe eines neuen Kontakts (Validierung notwendig) zu verwenden. Zudem wurden die Branches *ok* (Eingabe beendet), *cancel* (Eingabe abgebrochen) und *error* (s.o.) definiert.


A screenshot of a Java Swing dialog box titled "Kontakt bearbeiten". The dialog has a standard title bar with minimize, maximize, and close buttons. Inside, there are several text input fields arranged vertically, each preceded by a label: "Vorname:", "Nachname:", "Firma:", "Straße:", "PLZ:", "Stadt:", "Tel:", and "Email:". Below these is a dropdown menu labeled "Gruppe:" with "CUSTOMER" selected. At the bottom of the dialog are two buttons: "Abbrechen" and "Ok".

Abbildung 5: Kontaktdaten anzeigen und verändern

Zuletzt fehlt noch die Möglichkeit, dass der Nutzer einen Kontakt aus einer Liste von Kontakten auswählen kann, sprich auf das Ergebnis einer Suchanfrage reagieren kann. Das hier zu implementierende SIB benötigt also einen Parameter in Form einer Liste von `Contact`-Objekten. Auf der anderen Seite muss das `Contact`-Objekt zurückgeliefert werden, welches der Nutzer aus-

gewählt hat. Ähnlich zu vorherigen SIBs, sind diese Parameter als `ContextKey` realisiert. Das SIB liest die Kontaktliste aus dem Ausführungskontext und befüllt ein entsprechendes Fenster. Nachdem der Nutzer einen Kontakt ausgewählt hat, werden die Daten entsprechend in den Kontext geschrieben. Es wurden die Branches *ok* (Eingabe beendet), *cancel* (Eingabe abgebrochen) und *error* (s.o.) definiert.

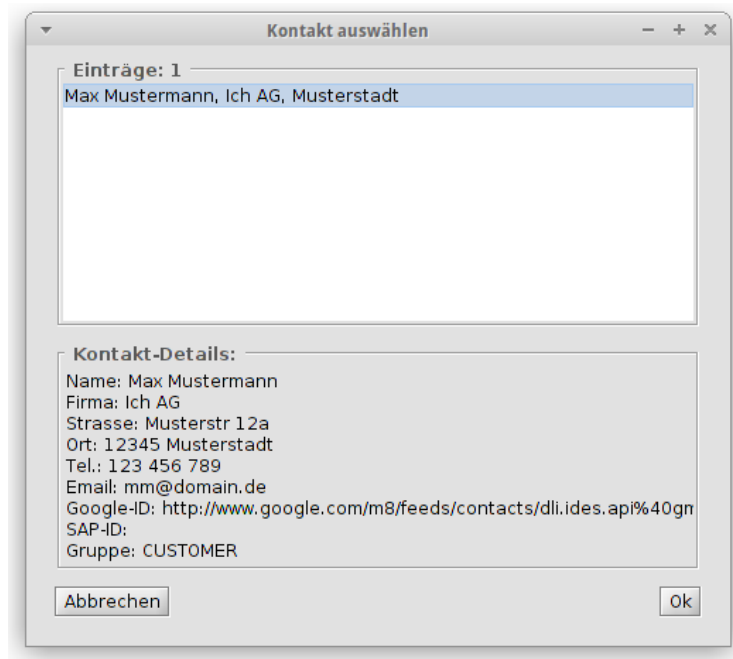


Abbildung 6: Einen Kontakt aus der Liste der Suchergebnisse auswählen

Um der Konvention von jABC zu folgen, wurde noch ein weiteres SIB implementiert. In jABC ist es üblich ein sogenanntes `Put`-SIB zu verwenden, wenn eine Variable eines bestimmten Typs im Ausführungskontext erzeugt wird. Zu diesem Zweck startet das Prozessmodell mit einem SIB namens `PutContact`. Es erzeugt eine Instanz der `Contact`-Klasse im Ausführungskontext, welche im Verlauf der Modellausführung verändert wird, bis ihr Inhalt zum Schluss dem Google-System hinzugefügt wird.

2.3.3 Besonderheit der GUI-SIBs

Eine Besonderheit bei der Implementierung von SIBs ist die Benutzung von Komponenten des Swing-Frameworks. Üblicherweise werden die erzeugten Fenster in einem separaten Thread gestartet, was den Effekt zur Folge hat, dass nach Erzeugung des Fensters die `trace()`-Methode nicht auf Eingaben wartet, sondern weiter ausgeführt wird. Während der Nutzer seine Eingabe noch nicht einmal begonnen hat, ist jABC mit der Ausführung der `trace()`-Methode schon fertig.

Als Lösung des Problems bot sich die Java-eigene Möglichkeit zur Thread-Synchronisation mittels eines `synchronized`-Blockes an. Eine exemplarische Verwendung zeigt Listing 10.

```
public String trace(ExecutionEnvironment env) {  
    ...  
    synchronized (frame) {  
        frame.wait();  
        ...  
    }  
}
```

Listing 10: Thread-Synchronisation in Java

In diesem Listing ist der `synchronized`-Block der `trace()`-Methode zu sehen. Nachdem der Swing-Frame erzeugt wurde, dient jener als Synchronisationsobjekt. Nachdem der `synchronized`-Block betreten wurde, wird in `trace()` die Methode `wait()` aufgerufen. An dieser Stelle stoppt die Ausführung von `trace()`, bis das entsprechende Gegenstück, die Methode `notify()`, auf dem Synchronisationsobjekt aufgerufen wurde. Dies geschieht innerhalb der `ActionListener` der verwendeten Buttons. Wenn der Nutzer mit der Eingabe der Daten oder der Auswahl des Kontakts fertig ist, signalisiert er dies mit einem Klick auf einen Button. Es wird der entsprechende `ActionListener` des Buttons aufgerufen, welcher wiederum innerhalb eines `synchronized`-Block die Methode `notify()` aufruft und anschließend das Fenster schließt.

Die wartende `trace()`-Methode wird daraufhin weiter ausgeführt und kann über geeignete Klassenvariablen des Frame-Objektes die Eingaben des Nutzers abfragen und in den Ausführungskontext schreiben.

2.3.4 Die JAVA-Laufzeitumgebung in jABC

Eine weitere Besonderheit hat sich während des Projektes zufällig ergeben. Um das Zusammenspiel von Konnektoren und GUI-Elementen effizient zu testen, wurde auf die Ausführung der Komponenten im jABC zunächst verzichtet. Erst zum Ende des Projektes wurden die Komponenten mittels ihrer zugehörigen SIBs in jABC getestet. Es zeigte sich ein Fehler während der Ausführung, welcher außerhalb von jABC nicht auftrat. Die geworfene `Exception` ließ vermuten, dass bestimmte Bibliotheken innerhalb der Ausführung von jABC nicht gefunden werden konnten. Unter Verwendung des selben Quellcodes außerhalb von jABC zeigte sich dieser Fehler jedoch nicht. Dies legt die Vermutung nahe, dass die zur Laufzeit existierende Java-Umgebung in Form der Java-Virtual-Maschine (JVM) in beiden Szenarien nicht gleich ist. Einige Bibliotheken werden innerhalb von jABC unter anderen Paketpfaden referenziert, als es außerhalb der Fall ist. Zu diesem Zweck mussten die Eigenschaften der JVM im jABC manuell verändert werden. Dies ist in Listing 11 zu sehen.

```
...
System.setProperty("javax.xml.parsers.SAXParser", ...);
System.setProperty("...parsers.SAXParserFactory", ...);
System.setProperty("oracle.xml.parser.v2.SAXParser", ...
...
```

Listing 11: Änderung von JVM-Eigenschaften

Auch wenn die Veränderung der Laufzeit-Eigenschaften das Problem behoben haben, so ist doch generell von dieser Praxis abzuraten. Denn die veränderten Eigenschaften bleiben auch noch nach der Modellausführung im jABC bestehen, da jABC selbst ein Java-Programm ist und zur Ausführung von Modell die JVM des Programms verwendet wird. Die Ausführung anderer Modelle im Anschluss an jenes dieses Projektes kann also beeinträchtigt oder im schlimmsten Fall gar nicht erst möglich sein.

3 Der fertige Prozess im jABC

Abschließend fließen nun die einzelnen Komponenten, welche durch entsprechende SIBs repräsentiert werden, in das Prozessmodell ein. Die folgende Abbildung 7 zeigt das fertige Projektergebnis, welches die in Kapitel 1.1 beschriebene Aufgabenstellung erfüllt.

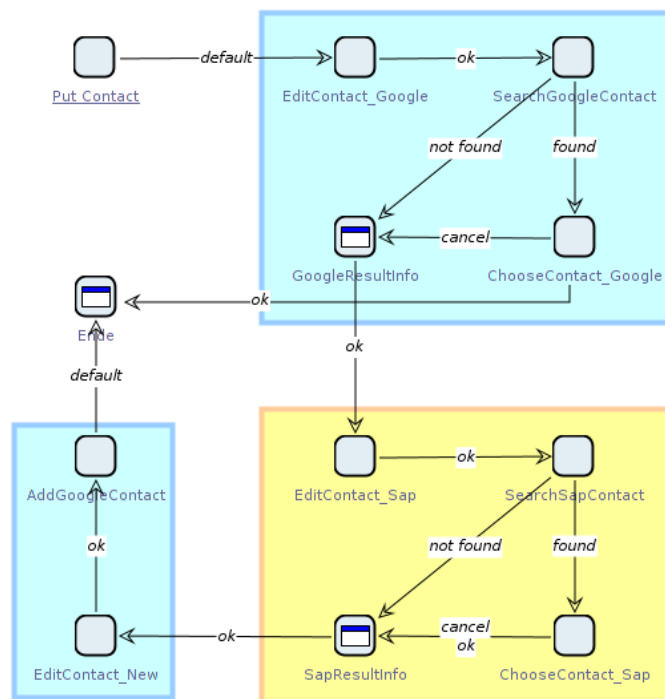


Abbildung 7: fertiges Prozessmodell im jABC

Zunächst lassen sich die farblich-gekennzeichneten Bereiche erkennen. Die blauen Bereiche enthalten die Kommunikation mit der Datenbank bei Google, während der gelbe Bereich die Kommunikation mit dem SAP-System repräsentiert. Die farbige Kennzeichnung dient auch der Tatsache, dass sich diese Bereiche in Untermodelle auslagern lassen, um diese z.B. in anderen Modellen wiederzuverwenden. Im Rahmen dieses Projektes wurde jedoch aus Gründen der Übersichtlichkeit darauf verzichtet. Ebenso verhält es sich mit den in Kapitel 2.3.2 beschriebenen *error*-Branches.

Zudem ist die Wiederverwendung der SIBs erkennbar, welche für die Nutzereingabe

verantwortlich sind und es fällt auf, dass die Teilgraphen der Suchanfragen bei SAP und Google eine gemeinsame Struktur haben.

4 Beurteilung des Projektes

Zusammenfassend lässt sich das Projektergebnis als Erfolg verzeichnen. Zum einen natürlich, weil das erzeugte jABC-Modell den Anforderungen der Aufgabenstellung genügt. Der Großteil des Erfolges ist jedoch sicherlich der Erfahrung geschuldet, welche während des Projektes gesammelt werden konnte. So zeigten sich während der Bearbeitung die Besonderheiten einer Datenmigration in Zusammenhang mit der Verwendung von Webservices.

So muss zum einen darauf geachtet werden, ob gewählte Webservices überhaupt die Methoden zu Verfügung stellen, welche für die Aufgabenstellung benötigt werden. Während es in diesem Fall zum Beispiel möglich war bei SAP nach einzelnen Attributen von Personen zu suchen, bot der Webservice von Google diese Möglichkeit nicht. Hier mussten bei jeder Anfrage alle Kontakte zurückgeliefert werden. Ein Aussortieren fand erst im Anschluss lokal statt.

Zum anderen ist darauf zu achten, dass im Zielsystem auch alle Attribute des Quellsystems gespeichert werden können. Unbedingt muss in diesem Zusammenhang auch darauf geachtet werden, eine ausreichende Zuordnung der Attribute des Quell- und Zielsystems zu spezifizieren. In einigen Fällen kann es daher vorkommen, dass sich bestimmte Informationen bei einer Migration nicht übertragen lassen. Im speziellen Fall dieses Projektes gab es hinsichtlich dieses Aspektes keine Probleme.

Nicht zuletzt kann das Projekt auch hinsichtlich der Zusammenarbeit der Projektteilnehmer als Erfolg verzeichnet werden.

5 Literaturverweise

Literatur

- [1] Google Developers; *Using OAuth 2.0 to Access Google APIs*; <https://developers.google.com/accounts/docs/OAuth2>; 29. Januar 2013
- [2] Google Code; *gdata Java Client*; <http://code.google.com/p/gdata-java-client/downloads/list>; 29. Januar 2013