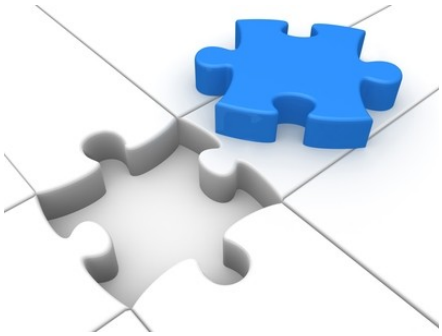
Der Einsatz von SOAP und WSDL am Beispiel von LIMBAS

# Ein Fall für zwei



Es gibt gute Gründe, warum Unternehmen heute eine Vielzahl von stark differenzierten Anwendungen einsetzen. Wenn jede einzelne Anwendung auf ihrem Gebiet spezialisiert ist, bringt dies in der Summe viele Vorteile. Auf der anderen Seite würde man sich natürlich auch eine Unternehmenssoftware wünschen, die alles am besten kann, auch wenn diese in der Realität nicht existiert. Denn häufig müssen verschiedene Anwendungen auf eine gemeinsame Datenbasis zurückgreifen oder gemeinsam übergreifende Geschäftsprozesse und Workflows umsetzen, was wiederum für einen hohen Integrationsaufwand verantwortlich ist.

Immerhin stellen Softwarehersteller - meistens über Middleware-Lösungen - Schnittstellen bereit, mit denen externe Anwendungen integriert werden können. Solche Schnittstellen werden heute häufig mit SOAP realisiert. In Verbindung mit SOAP hat sich der Standard WSDL etabliert.

In diesem Artikel wird die Funktionsweise von WSDL und SOAP am Beispiel von LIMBAS gezeigt. LIMBAS ist ein Open-Source-PHP-Framework [**1**], das für die Umsetzung von web-basierten Datenbankanwendungen, wie z.B. CRM, ERP, DMS geeignet ist.

Seit Version 2.6 ermöglicht es Limbas, über die WSDL-Schnittstelle grundlegende Datenbankoperationen durchzuführen.

Das für Demonstrationszwecke gedachte LIMBAS CRM-System (**Abb.1**) ist auf der Live-CD, VM-Version und dem Installationspaket enthalten. Dieses ist mit vielen Beispiel-Datensätzen gefüllt und daher gut geeignet, um Datenbankoperationen auf der WSDL-Schnittstelle mit Hilfe von SOAP-Clients durchzuführen. Dafür sind keine tiefergehende Programmierkenntnisse notwendig. Abschließend wird gezeigt, wie über die Datenintegration von dem LIMBAS CRM in Google Docs ein reales Anwendungsszenario für die WSDL-Schnittstelle umgesetzt werden kann.

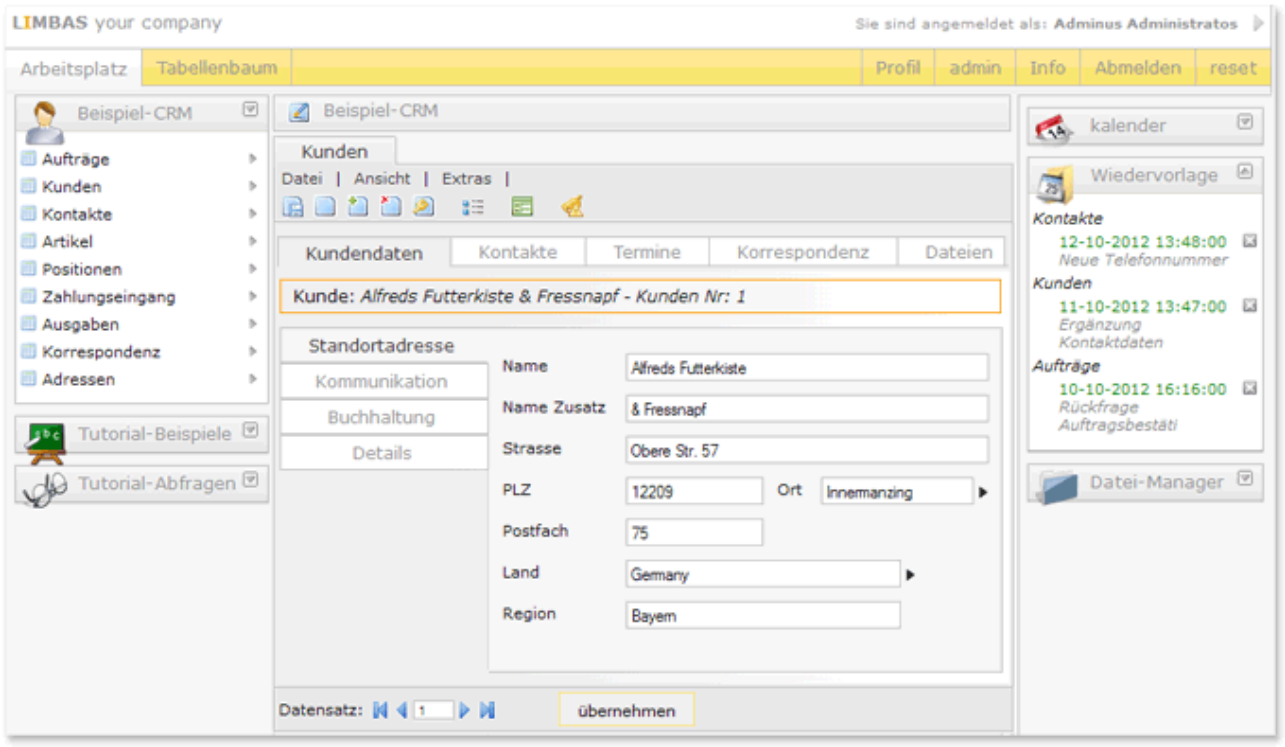



*Abb. 1: Das LIMBAS CRM*

# WSDL

Die Web Service Description Language (WSDL) ist im Wesentlichen die Spezifikation eines XML-Schemas, das dazu gedacht ist, externe Schnittstellen von Anwendungen in XML zu definieren.

Ein WSDL-Dokument enthält dazu die folgenden Informationen:

- *Ports / Endpoints*: Adresse (URL) des Webservice, der die Schnittstelle bereitstellt.
- *Bindings*: Funktionen, die ein Port besitzt und das Protokoll, das dieser einsetzt (i.d.R. SOAP).
- *Messages und (Complex) Types*: Aufbau von Nachrichten zur Initiierung von Funktionsaufrufen und Definition der Parameter- und Rückgabewerttypen.

LIMBAS stellt für jede Datenbanktabelle eine separate, dynamisch erzeugte WSDL-Schnittstelle bereit. Diese folgen einem gängigem URL-Schema. So steht z.B. das WSDL-Dokument für die Tabelle Kunden unter der URL http://localhost/limbas/dependent/main_wsdl.php?Service=Kunden&WSDL bereit. Hierbei ist zu beachten, dass der Tabellenname mit großem Anfangsbuchstaben und im Folgenden klein geschrieben wird.

Aus der Port-Definition der Kunden-WSDL ergibt sich die URL der Webservice-Schnittstelle wie folgt:

```
<wsdl:port name="KundenPort" binding="tns:KundenBinding">
<soap:address location="https://limbasurl/dependent/main_wsdl.php?Service=Kunden"/>
</wsdl:port>
```

Desweiteren kann aus dem Binding entnommen werden, dass die WSDL-Schnittstelle als Protokoll SOAP verwendet:

```
<wsdl:binding name="KundenBinding" type="tns:KundenPortType">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="query">
        ...
</wsdl:operation>
<wsdl:operation name="getByPk">
        ...
</wsdl:operation>
<wsdl:operation name="delete">
        ...
</wsdl:operation>
<wsdl:operation name="insert">
        ...
</wsdl:operation>
<wsdl:operation name="update">
        ...
</wsdl:operation>
<wsdl:operation name="join">
        ...
</wsdl:operation>
</wsdl:binding>
```

An dieser Stelle sollte noch erwähnt werden, das für SOAP verschiedene Styles existieren, die festlegen, wie ein Request aufgebaut sein muss (siehe Abschnitt „SOAP“). LIMBAS verwendet den RPC-Style, da dieser für den Entwickler am einfachsten zu handhaben ist.

Die LIMBAS WSDL-Schnittstelle stellt die Methoden *query*, *getByPk*, *delete*, *insert*, *update* und *join* bereit: Diese Funktionen ermöglichen das Abfragen, Löschen, Erstellen, Bearbeiten und Verknüpfen von Datensätzen in Limbas.

Ein besonderer Vorteil von WSDL ist die Möglichkeit für Funktionen auch komplexe Typen als Parameter- und Rückgabewerte zu definieren, wie sie z.B. in der objektorientierten Programmierung oder für Datenbankschemas verwendet werden. Betrachten wir nun die Request und Response-Nachrichten, die von der Query-Funktion entgegengenommen bzw. zurückgegeben werden:

```
<wsdl:message name="queryRequest">
<wsdl:part name="params" type="tns:KundenQuery"/>
</wsdl:message>
<wsdl:message name="queryResponse">
<wsdl:part name="return" type="tns:KundenArray"/>
</wsdl:message>
```

Die Parameter eines *queryRequest* besitzen den Typ „*KundenQuery*“. Dieser ist wie folgt aufgebaut:

```
<wsdl:types>
<xsd:complexType name="KundenQuery">
<xsd:sequence>
<xsd:element name="q_orderby" type="xsd:string"/>
<xsd:element name="q_page" type="xsd:string"/>
<xsd:element name="q_rows" type="xsd:string"/>
<xsd:element name="q_nolimit" type="xsd:string"/>
<xsd:element name="ID" type="xsd:string"/>
<xsd:element name="NAME" type="xsd:string"/>
<xsd:element name="NAME_ZUSATZ" type="xsd:string"/>
<xsd:element name="STRASSE" type="xsd:string"/>
<xsd:element name="ORT" type="xsd:string"/>
<xsd:element name="PLZ" type="xsd:string"/>
<xsd:element name="LAND" type="xsd:string"/>
                              ...
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="...">
                            ...
</xsd:complexType>
</wsdl:types>
```

Eine Abfrage kann sich demnach auf beliebig viele Attribute der Kunden-Tabellen beziehen (siehe auch Abschnitt „SOAP“).

Außerdem kann das Ergebnis der Abfrage in LIMBAS über folgende Parameter angepasst werden:

- *q_orderby*: Die Datensätze können nach dem hier angegebenen Element-Namen sortiert werden.
- *q_rows*: Gibt die Anzahl der Datensätze an, die pro Seite ausgegeben werden sollen. Standardmäßig sind dies 30.
- *q_page*: Gibt die Seite/Seitenzahl der Datensätze an, die LIMBAS ausgeben soll.
- *q_nolimit*: Falls dieser Parameter auf 1 gesetzt ist, gibt es keine Beschränkung in der Anzahl der abgefragten Datensätze.

Das Ergebnis der *query*-Funktion besitzt den Typ *KundenArray*. Dieser enthält die Elemente des Datensatzes.

Die LIMBAS Rechteverwaltung ist vollständig in die WSDL-Schnittstelle integriert. Als Folge sind für einen Benutzer immer nur die Elemente eines Datensatzes sichtbar, auf die er autorisiert ist zuzugreifen. Entsprechend der Zugriffsrechte werden im WSDL-Dokument auch die Typen *KundenQuery* und *KundenArray* dynamisch erstellt.

# SOAP

Mit SOAP können „Remote-Procedure-Calls“ über bestehende Netzwerkprotokolle wie z.B. HTTP durchgeführt werden. Der Standard basiert auf XML und ist deshalb offen, erweiterbar und plattformunabhängig. Aus diesem Grund wird SOAP häufig für die Umsetzung von externen Programm-Schnittstellen eingesetzt.

Mit dem Tool SoapUI [**2**] (**Abb.2**) können SOAP-Requests für die LIMBAS WSDL-Schnittstelle auf einfache Weise generiert und ausgeführt werden. Dieses wird benötigt, um die folgenden Schritte durchzuführen.

Für die Einrichtung wird hierzu in SoapUI ein neues Projekt angelegt und das WSDL-Dokument für die Kundentabelle hinzugefügt . Um eine Datenabfrage umzusetzen, kann dann im *Kundenbinding* die Funktion *query* ausgewählt und mit Rechtsklick ein neuer Request erstellt werden. Der SOAP-Request enthält nun ein *Body*-Element, in dem die Abfrageparameter angegeben werden:

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:Kundenwsdl">
<soapenv:Header/>
<soapenv:Body>
<urn:query soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<params xsi:type="urn:KundenQuery">
<q_orderby xsi:type="xsd:string">?</q_orderby>
<q_page xsi:type="xsd:string">?</q_page>
<q_rows xsi:type="xsd:string">?</q_rows>
<q_nolimit xsi:type="xsd:string">?</q_nolimit>
<ID xsi:type="xsd:string">?</ID>
<NAME xsi:type="xsd:string">?</NAME>
<NAME_ZUSATZ xsi:type="xsd:string">?</NAME_ZUSATZ>
<STRASSE xsi:type="xsd:string">?</STRASSE>
<ORT xsi:type="xsd:string">?</ORT>
<PLZ xsi:type="xsd:string">?</PLZ>
<LAND xsi:type="xsd:string">?</LAND>
                ...
</params>
</urn:query>
</soapenv:Body>
</soapenv:Envelope>
```

Für jedes Attribut kann der Platzhalter „?“ durch einen Abfragewert ersetzt werden. Es ist auch möglich, Elemente aus der *Query* auszulassen.

So wird z.B. eine Abfrage aller Kunden in Mannheim folgendermaßen beschrieben:

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"xmlns:soapenv="http://schemas.xmlsoap.org/soap
/envelope/" xmlns:urn="urn:Kundenwsdl">
<soapenv:Header/>
<soapenv:Body>
```

<urn:query soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<params xsi:type="urn:KundenQuery">
<ORT xsi:type="xsd:string">Mannheim</ORT>
</params>
</urn:query>
</soapenv:Body>
</soapenv:Envelope>

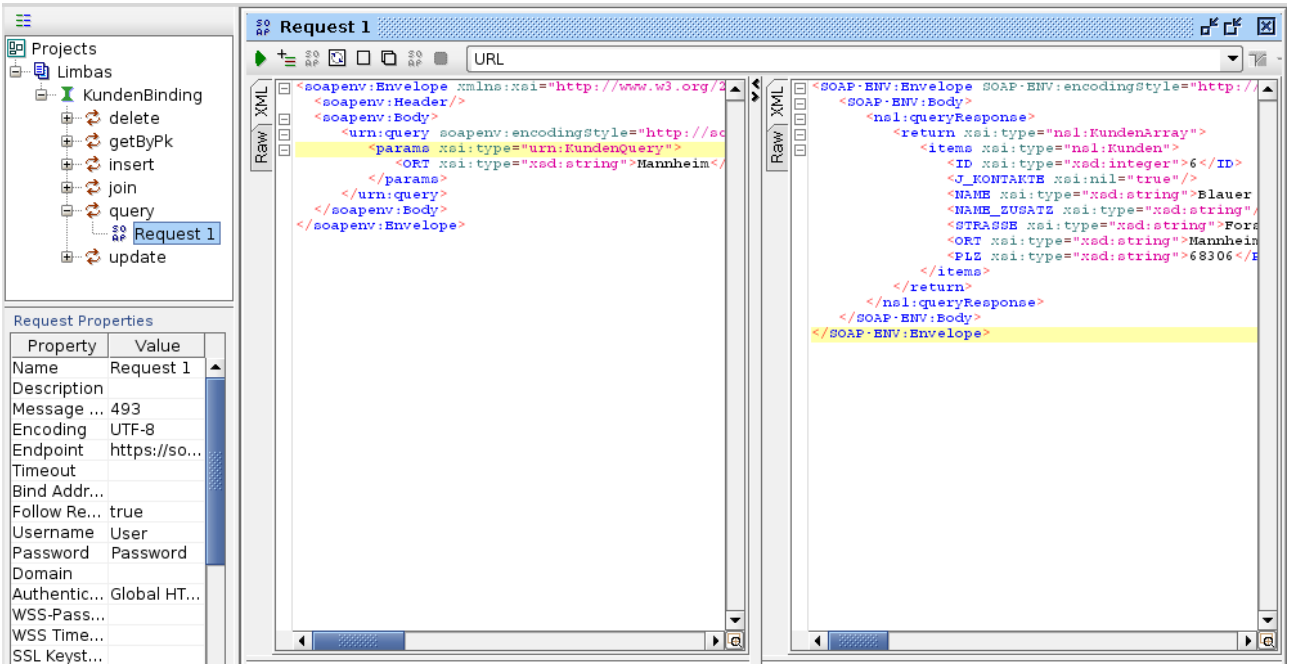



*Abb. 2: SoapUi*

Bei der weit verbreiteten Kombination SOAP über HTTP (welche auch LIMBAS einsetzt) wird der in XML beschriebene Funktionsaufruf in einem HTTP-POST-Request an eine URL gesendet und von einem Web- oder Applikationsserver entgegengenommen.

**Hinweis:**
LIMBAS setzt HTTP-Authentifizierung für SOAP-Requests voraus. In SoapUi müssen deswegen im Bereich „Request-Properties“ die Felder *Username* und *Password* gesetzt werden.

Auf den vorhergehenden Request zur Kundenabfrage würde LIMBAS alle gefundenen Datensätze in XML wie folgt zurückgeben:

<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns1="urn:Kundenwsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<ns1:queryResponse>
<return xsi:type="ns1:KundenArray">
<items xsi:type="ns1:Kunden">
<ID xsi:type="xsd:integer">6</ID>
<NAME xsi:type="xsd:string">Blauer See Delikatessen</NAME>

```
<NAME_ZUSATZ xsi:type="xsd:string"/>
<STRASSE xsi:type="xsd:string">Forsterstr. 57</STRASSE>
<ORT xsi:type="xsd:string">Mannheim</ORT>
<PLZ xsi:type="xsd:string">68306</PLZ>
                                        ...
</items>
</return>
</ns1:queryResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

LIMBAS unterstützt auch Platzhalter in Abfragen. Hierzu kann das Zeichen „*" als Wildcard an den Anfang und/oder an das Ende eines Suchbegriffs gestellt werden.

Außerdem können in LIMBAS die beiden logischen Operatoren *UND* und *ODER* jeweils mit den Zeichen „&&" und „||" verwendet werden. Das *UND* liefert die Schnittmenge der Datensätze, die alle Einschränkungen gleichzeitig erfüllen. Das *ODER* liefert im Gegensatz dazu die Viereinigungsmenge aller Datensätze, die zumindest jeweils eine der Einschränkungen erfüllen.

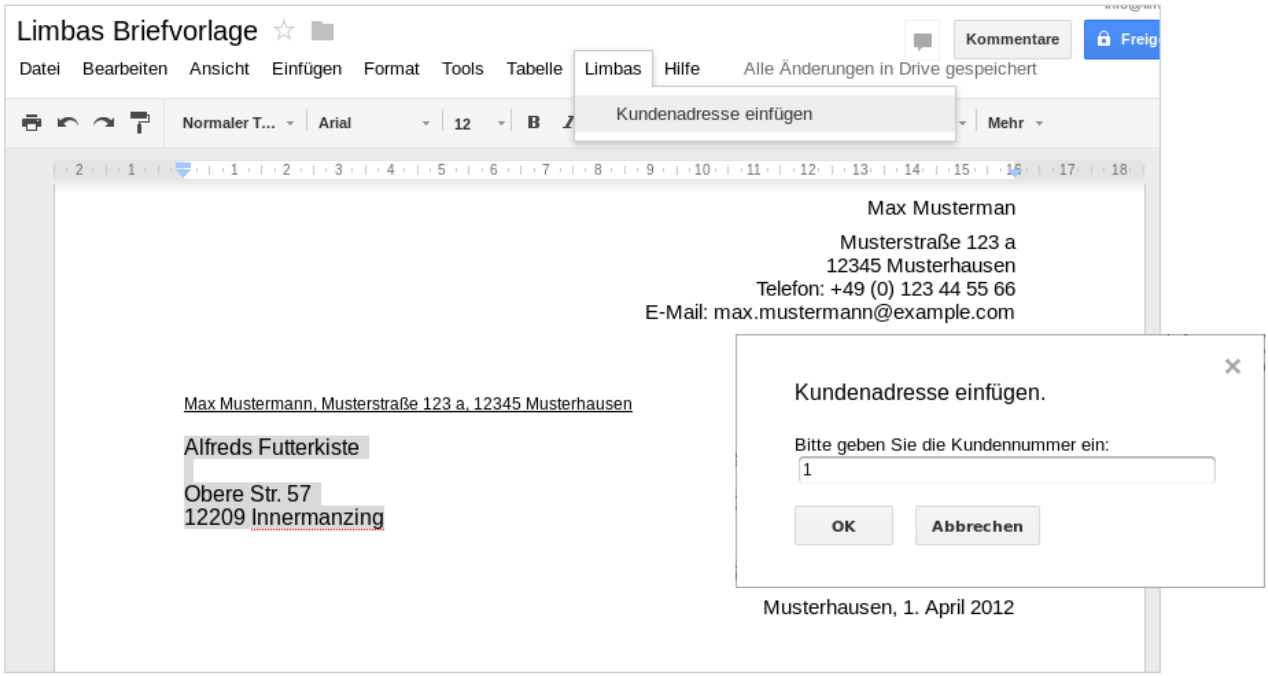
Wildcards und logische Operatoren sind im Allgemeinen für komplexere Abfragen notwendig. So können z.B. alle Kunden in den PLZ-Bereichen 6xxxx, 7xxxx, 8xxxx, 9xxxx mit dem Anfangsbuchstaben L und nach ihrem Name sortiert, folgendermaßen abgefragt werden:

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:Kundenwsdl">
<soapenv:Header/>
<soapenv:Body>
<urn:query soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<params xsi:type="urn:KundenQuery">
<q_orderby xsi:type="xsd:string">NAME</q_orderby>
<PLZ xsi:type="xsd:string"><![CDATA[6* || 7* || 8* || 9*]]></PLZ>
<NAME xsi:type="xsd:string">L*</NAME>
</params>
</urn:query>
</soapenv:Body>
</soapenv:Envelope>
```

# Google Docs Datenintegration von LIMBAS mit App Scripts und WSDL/SOAP

Für die LIMBAS WSDL-Schnittstelle gäbe es denkbar viele Anwendungsfällen. Ein möglichst einfacher wäre der folgende: Will man einen Brief an einen Kunden senden und hat nur seine Kundennummer zu Hand, dann wäre es praktisch,direkt aus der Textverarbeitung mit Hilfe eines Skript die Adresse automatisch in das Dokument einzufügen. In diesem Fall soll Google Docs als Beispiel eine LIMBAS Daten-Integration über WSDL dienen (**Abb.3**).

*Abb 3: LIMBAS in Google Docs*




Für Google Docs können eigene Erweiterungen in Google Apps Script - einer auf JavaScript basierenden Programmierprache - implementiert werden. Die Besonderheit dabei ist, dass Skripte serverseitig ausgeführt werden. Als externes System kann Google Docs auf die LIMBAS Webservice-Schnittstelle mit SOAP/HTTP-Post zugreifen.

Für viele Programmiersprache stehen Bibliotheken zur Verfügung, mit denen der Zugriff auf eine Webservice-Schnittstelle vereinfacht werden kann. Dies ist leider bei Google Apps Script nicht der Fall. Da aber bereits im Abschnitt SOAP gezeigt wurde, wie ein SOAP-Request aufgebaut ist und mit Hilfe von SoapUi erzeugt werden kann, kann auch ein einfacher Client für die Abfrage selbst entwickelt werden.

Bei Google-Docs können Skripte mit Hilfe des eingebauten Skripteditor entwickelt werden (im Kontextmenü unter Tools → Skripteditor). Es ist auch möglich, die Skripte über eigene Kontextmenüs und Formulare in die Benutzerschnittstelle einzubinden.

Das Google-Doc mit dem fertigen Skript wird unter [**3**] bereitgestellt.

Im folgenden wird die Funktion zur Abfrage der Kundendaten über die WSDL-Schnittstelle gezeigt. Für die Umsetzung der Benutzerschnittstelle wird auf den vollständigen Quellcode des Google-Doc verwiesen, der unter der zuvor genannten URL eingesehen werden kann.

```
function getCustomerAdress(kundennummer){
//SOAP Request konstruieren
var url="https://demo01.limbas.com/dependent/main_wsdl.php?Service=Kunden";
var body='<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:urn="urn:Kundenwsdl">'+
```

```
'<soapenv:Header/>'+'<soapenv:Body>'+
'<urn:getByPk soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">'+
'<params xsi:type="xsd:long">'+kundennummer+'</params>'+
'</urn:getByPk>'+
'</soapenv:Body>'+
'</soapenv:Envelope>';
var httpUser="limbasUser";
var httpPassword="limbasPassword";
var options =
 {
"headers" : {"Authorization": "Basic " + Utilities.base64Encode(httpUser + ":" + httpPassword),
"Content-Type": "text/xml", "SOAPAction": "urn:Kundenwsdl#getByPk", "charset": "UTF-8"},
"payload" : body,
 "method" : "post",
 };
//SOAP Request senden
var response = UrlFetchApp.fetch(url, options);
```

In den bereits zuvor für die *KundenQuery* generierten SOAP-Body wird die aus der Eingabemaske entnommene Kundennummer eingefügt. Dieser SOAP-Body wird dann als Inhalt des HTTP-Post-Requests mit der in Google App Scripts vorhandenen Funktion *UrlFetchApp.fetch* abgesendet. Dazu müssen nur noch die notwendigen HTTP-Header für SOAP und die bei LIMBAS notwendige HTTP-Authentifizierung ergänzt werden. Die Variable *url* ist spezifisch für eine Limbas-Installation und wird durch die im WSDL-Dokument unter dem Port des Kundenbindings angegebene Url der SOAP-Schnittstelle ersetzt. Desweiteren müssen die Variablen *httpUser* und *httpPassword* mit den Benutzerdaten von LIMBAS gesetzt werden. Ansonsten ist der Client auch schon fast fertig. Das Simple-Object-Access-Protokoll (SOAP) wird auf jeden Fall seinem Namen gerechnet. Der SOAP-Response muss nun nur noch mit Hilfe eines XML-Parsers ausgewertet werden:

```
// SOAP Response parsen
var document = XmlService.parse(response.getContentText());
var root = document.getRootElement();
var kunde = root.getChild("Body",
XmlService.getNamespace("http://schemas.xmlsoap.org/soap/envelope/")).getChild("getByPkRespo
nse", XmlService.getNamespace("ns1","urn:Kundenwsdl")).getChild("return").getChild("items");
var name=kunde.getChild("NAME").getValue();
var strasse=kunde.getChild("STRASSE").getValue();
 var plz=kunde.getChild("PLZ").getValue();
var ort=kunde.getChild("ORT").getValue();
```

Nachdem die einzelnen Adress-Bestandteile aus dem SOAP-Response entnommen wurden, können diese wieder zusammengesetzt und in das aktive Dokument geschrieben werden:

```
//Kundenadresse in aktives Dokument schreiben
  var doc = DocumentApp.getActiveDocument();
var cursor=doc.getCursor();
if (cursor)
cursor.insertText(name + "\n" + "\n" + strasse +"\n" + plz + " " + ort + "\n" );
```

## Fazit

Mit SOAP ist es möglich, verschiedene Anwendungen über Schnittstellen miteinander zu integrieren und das unabhängig davon, welche Technologien und Programmiersprachen sie verwenden, auf welchen Plattformen sie laufen und ob diese im lokalen Netzwerk oder über das Internet kommunizieren. WSDL vereinfacht unter anderem die Umsetzung von Clients, indem es z.B. die Generierung von SOAP-Requests ermöglicht. Da WSDL und SOAP auf XML basieren, sind sie in Kombination sehr gut geeignet, um komplex strukturierte Daten wie Objekte oder Dokumente auszutauschen. Nicht zuletzt durch die starke Anlehnung von XML-Schema an die objektorientierte Programmierung sind entsprechende Tools (wie Apache CXF) in der Lage, auf Basis eines WSDL-Dokuments den Quellcode für komplexe Server- und Client-Schnittstellen in verschiedenen Programmiersprachen automatisch zu generieren und dadurch dem Entwickler viel Arbeit und Zeit zu übernehmen.

LIMBAS bietet nicht nur das Front-End, um große Datenbestände zu verwalten, sondern ermöglicht externen Anwendungen auch den Zugriff auf Daten über dynamische WSDL-Schnittstellen. Ein weiterer Vorteil für Entwickler ist, dass diese im Vergleich zu SQL wesentlich einfacher zu handhaben sind Auch die Rechteverwaltung wurde hier integriert, sodass sichergestellt ist, das Benutzer von außen nur nach Authentifizierung auf die Datenbanktabellen und die Elemente von Datensätzen zugreifen können, für die sie autorisiert wurden.

Für den Aufruf von verschiedenen Systemfunktionen, die für eine Prozessintegration notwendig wären, stellt LIMBAS derzeit zusätzlich noch eine umfangreiche SOAP-Schnittstelle bereit. In Zukunft soll diese Schnittstelle auch über WSDL standardisiert werden.

---

Autor: Armin Litzel, TU-Student, arbeitet als Entwickler bei der LIMBAS GmbH.

---

### Links & Literatur

[**1**] http://www.limbas.com

[**2**] http://www.soapui.org/

[**3**] goo.gl/8eqma9