Inhaltsverzeichnis

1. PHP

Für die meisten Small Business Anwendungen bietet die Scriptsprache PHP (*Hypertext Preprocessor*) eine echte Alternative zu den größeren und auch wesentlich umfangreicheren Application Frameworks wie J2EE (Sun Microsystems) und .NET Framework (Microsoft). Zwar sind die Implementierungen von Standards wie SOAP und WSDL (XML Schema) bei den „großen Brüdern“ um einiges komplexer/vollständiger[1] und die Integrierung in die bestehend Application Infrastructure sollte keinen großen Mehraufwand mit sich bringen, jedoch sind solch teure und vor allem aufwendige Lösungen nicht für jedes Unternehmen rentabel. Hier kommt PHP ins Spiel:

1.1. PHP und Web Services

Allgemeines zu PHP:
PHP ist eine *der* Einstiegssprachen in das Programmieren (eigene Erfahrung) und kann – meiner Meinung nach – als diese auch jedem weiter empfohlen werden. Es werden mächtige Sprachkonzepte wie Klassen, die das objektorientierte Programmieren ermöglichen, unterstützt. Zudem wird der Programmierer von unangenehmen Aufgaben wie Speicherverwaltung und Typ-Konvertierung (`integer` → `string`) ferngehalten.

Einsatzgebiete von PHP sind meistens Server-seitige Scripts, um dynamischen Inhalt zu erzeugen. So gibt es heutzutage schon eine Reihe von Content Management Systemen mit Web Frontend, die rein in PHP programmiert worden sind (z.B. Typo3). Und genau für derartige Anwendungsszenarien ist die Sprache auch ausgelegt: Header zum Versandt von Dokumenten über das Netz werden automatisch erzeugt und den Interpreter gibt es schon seit längerer Zeit als Modul-Version für den Apache Web Server. Die Verwendung von PHP in Verbindung mit HTTP und WWW liegt somit nahe. Der Vollständigkeit halber sei aber noch erwähnt, dass man die Sprache auch in ganz anderen Gebieten einsetzten kann. So bietet das Paket PHP-Gtk zum Beispiel die Möglichkeit, GUI-basierte Programme zu entwickeln.

Viele dieser Pakete kann man im PHP-eigenen Repository namens PEAR finden. Wer sich damit näher auseinander setzen möchte, sei auf die Seite http://pear.php.net verwiesen.

1 http://www.scottnichol.com/nusoapprog.htm

Der Einsatz von PHP hat aber auch noch wirtschaftliche Vorteile: seitdem die Scriptsprache seit der Version 4 so unglaublich rasant an Popularität in der Internet Gemeinschaft gewonnen hat, bieten die meisten Provider Web Server mit vorinstallierter PHP-Unterstützung an, und das schon zu geringen Preisen. Die Wartung des Systems, welche bei vergleichbaren J2EE oder .NET Installationen um einiges aufwendiger und damit auch kostspieliger wäre, wird dabei natürlich vom Provider übernommen. Auch die Wartung der Web Anwendungen sollte im Small Business Bereich in Verbindung mit PHP in aller Regel die ökonomischere Variante sein, wohingegen zum Beispiel .NET in Verbindung mit einer komplexen Application Infrastructure die vorteilhaftere Wahl wäre.

Natürlich hat die Sprache auch Nachteile:
Der wohl wichtigste ist die Tatsache, dass PHP „nur" eine Scriptsprache ist, und deshalb im Hinblick auf z.B. die Geschwindigkeit der Ausführung den „großen Brüdern" keine ernsthafte Konkurrenz bietet. Künftig wird dies aber immer weniger eine Rolle spielen aufgrund der rasanten Entwicklung von Hard- und Software.

Zusammenfassend kann man wohl folgende Regel aufstellen:
PHP für Small Business, J2EE oder .NET für eine komplexe Application Infrastructure.

1.1.1. SOAP-APIs in PHP

Es stehen verschiedene SOAP-Bibliotheken für PHP zur Verfügung. Die wichtigsten sind in der nachfolgenden Auflistung[2] abgebildet:

- **SOAPx4**
  Diese Klassensammlung ist die Mutter einiger populärer SOAP-Implementierungen, wird aber in ihrer ursprünglichen Form nicht mehr verwendet.
  - **NuSOAP**
    Bei NuSOAP handelt es sich um die Weiterentwicklung der Bibliothek SOAPx4 durch den ehemaligen Gründer: Dietrich Alaya. Zur Zeit wird sie wieder aktiv weiterentwickelt hat deshalb gute Zukunftschancen.
  - **PEAR::SOAP**
    Auch das PHP Repository enthält ein Paket für SOAP: PEAR::SOAP. Vorteile sind die Anpassung der SOAPx4-Klassen an PEAR-Standards (z.B. die Verwendung von PEAR-Errors zur Repräsentation von Fehlern) und die große Popularität sowie aktive Weiterentwicklung der Bibliothek.
  - **SWSAPI**
    Der Vertreiber des allseits bekannten Perl-Interpreters – Active State – bietet eine SOAP-Implementierung für PHP an. Ein großer Vorteil ist die zusätzliche Verfügbarkeit der Bibliothek in Perl und Python. Da einer der Entwickler aber ebenfalls an PEAR::SOAP arbeitet, wird im Allgemeinen die Verwendung des PEAR-Pakets empfohlen.
- **PHP-SOAP**
  Es gibt natürlich bereits eine C-Implementierung von SOAP: PHP-SOAP. Offensichtlich ist *der* große Vorteil der Bibliothek die Geschwindigkeit, leider gilt sie aber zur Zeit noch als instabil: trotz alledem ein heißer Kandidat auf die offizielle SOAP-Erweiterung von PHP.
- **Krysalis SOAP**
  Krysalis SOAP ist eine SOAP-Erweiterung für das Krysalis Framework und kann auch nur in Verbindung mit diesem wirklich ökonomisch eingesetzt werden. Deshalb ist diese Bibliothek nur für Krysalis-Nutzer wirklich interessant.
- **eZ SOAP**
  Ebenfalls ein Nischendasein fristet die SOAP-Bibliothek für das Content Management System eZ Publish und ist somit auch nur für eZ Publish-Nutzer sinnvoll.

2 Web Services mit PHP (Galileo Computing): *3.1.4. SOAP-Implementierungen für PHP*

Welche dieser Bibliotheken im Realfall nun wirklich eingesetzt werden soll, kann ich nicht sagen. Im Rahmen meiner Seminararbeit habe ich nur mit einer Bibliothek – NuSOAP – gearbeitet und diese war im Allgemeinen wirklich einfach zu verstehen und gut Hand zu haben. Wie es aber so oft in der Softwareentwicklung der Fall ist, lässt sich die eine Bibliothek besser in das System integrieren, als eine zweite, die denselben Funktionsumfang bietet. Entscheiden Sie also selbst!

In den folgenden Abschnitten wollen wir uns an zwei Beispielen die Entwicklung von Clients für bereits bestehende Web Services sowie die Implementierung eines eigenen Service unter Verwendung der Bibliothek NuSOAP ansehen. Der Umstieg auf andere SOAP-Erweiterungen, die von SOAPx4 abstammen, sollte aufgrund der großen Gemeinsamkeiten bezüglich ihrer Struktur und Verwendung relativ leicht fallen.

1.1.2. NuSOAP im Überblick

Das Projekt NuSOAP (aktuell in der Version 0.6.7) wird von Dietrich Alaya betreut, der auch schon als Gründer und Hauptentwickler an der Bibliothek SOAPx4 beteiligt war (NuSOAP ist sozusagen die neueste Version). Der Name ist auf den Sponsor NuSphere (http://www.nusphere.de) zurückzuführen, der für für die Entwicklungsumgebung PHP-Ed bekannt ist. Die offizielle Homepage des Projekts ist unter http://dietrich.ganx4.com/nusoap/ zu finden.

Technische Details:
NuSOAP unterstützt offene Standards wie SOAP 1.1, WSDL 1.1 und HTTP 1.0/1.1. SOAP-Nachrichten werden also standardmäßig über HTTP versandt und empfangen. Es lassen sich aber auch auf einfache Weise Protokolle wie SMTP oder HTTPS (mit *curl*) auf Client-Seite nutzen. Um dem gegenüber einen Service zu entwickeln, der Anfragen über andere Protokolle erhält, ist etwas mehr Aufwand notwendig, da nicht mehr der Web Server (HTTP Server) für die Verarbeitung von diesen Nachrichten zuständig ist. Das kann aber leider im Rahmen dieses Dokuments nicht demonstriert werden.

Laut Aussage des Hauptentwicklers – Scott Nichol – ist die Bibliothek NuSOAP nicht so komplett im Funktionsumfang wie die vergleichbaren Produkte für .NET und Java (Apache Axis). Es lassen sich aber viele wichtige Dinge – wie die von Java RMI bekannte Serialisierung/Encodierung (Marshalling) und Deserialisierung/Decodierung (Unmarshalling) von sprachinternen Datentypen – auf einfache Weise realisieren (*SOAP encoding*).

Für Informationen bezüglich der Programmierung mit NuSOAP gibt es mehrere Quellen: Auf der Homepage wird eine API-Dokumentation angeboten, die aber meiner Meinung nach nicht wirklich hilfreich ist (das Urteil bleibt aber jedem selbst überlassen). Eine bessere Alternative bieten die zahlreichen Tutorials, von denen einige ausgewählte auch auf der Projekt-Webseite verlinkt sind. Weitere lassen sich, wie in den meisten Fällen, auch mit *google* finden. Wenn man gar nicht mehr voran kommt, kann man sich auch den Quelltext (aktuell 5776 Zeilen mit Kommentaren und Whitespaces) der Bibliothek ansehen. Mit ausreichend Erfahrung und Programmierkenntnissen in PHP sollte es keine allzu große Herausforderung darstellen, den gut kommentierten Code zu verstehen und die Informationen zu verwenden.

Wie man ohne weiteres schnell bemerken wird, ist die Bibliothek mit Klassen als zugrunde liegendem Konzept realisiert. Somit ist eine Verwendung von Services und Clients als Objekten möglich. Wie das im Detail funktioniert, wird ausschnittsweise in den folgenden Abschnitten demonstriert.

1.2. Konsumieren eines Web Service

Um einen Web Service zu verwenden, nützt vielleicht folgende Betrachtungsweise: ein Web Service verhält sich wie ein Schnittstelle (in Java: `interface`). Um den Web Service zu verwenden, muss man sich also die Schnittstelle/das Interface ansehen. Hat man zum Beispiel ein Objekt einer Grafik-Schnittstelle, so kann man Methoden wie das Zeichnen von Objekten oder Registrieren von Interaktionsabläufen mit dem Benutzer zur Verfügung. Diese haben ihrerseits wieder eine ganz

spezielle Aufrufsemantik. Dazu gehören die Informationen, welche Parameter den Methoden übergeben werden müssen und was die Rückgabewerte sind (dies wird über Typ-Informationen angegeben).

Genauso verhält es sich bei den Web Services. Es werden verschiedene Operationen bereitgestellt, die ihrerseits wiederum mit `input`- und/oder `output`-Nachrichten verknüpft sind (dass hier noch die Reihenfolge eine Rolle spielt, soll die Betrachtung nicht weiter stören, denn trotz alledem wird eine Aufrufsemantik definiert). Diese Nachrichten entsprechen den Parametern und Rückgabewerten beim Methodenaufruf in Java.

So stellen sich also primär zwei Fragen:

- Welche Operationen werden bereitgestellt?
- Wie verwendet man diese Operationen?
  (damit ist gemeint, welche Aufrufsemantik die einzelnen Operationen haben)

Wenn man sich dann für eine Bibliothek in einer beliebigen Programmiersprache entschieden hat, sollte man sich noch mit folgenden Dingen auseinander setzten:

- Wie sende ich eine Anfrage?
- Wie werte ich das Ergebnis aus?

Wie man diese Fragen nun in der Praxis beantwortet und wo man die Antworten auf die Fragen findet, wird im Folgenden am Beispiel eines Clients für den *google Web Service* gezeigt.

1.2.1. Welche Operationen stellt der Web Service bereit?

Im Wesentlichen gibt es zwei Hauptquellen für die Informationen:

- die *WSDL-Datei*
  Die meisten Anbieter von Web Services veröffentlichen eine XML-Datei, welche Informationen über den Service enthält. Da die Datei in einer Meta-Sprache geschrieben ist, kann sie sowohl von Menschen, als auch von Maschinen gelesen und verstanden werden. Jedoch wird in machen Fällen nicht vollständig klar, wie z.B. Operationen verwendet werden. Dazu werden wir aber gleich ein Beispiel kennen lernen.
- die *API-Dokumentation*
  Die Alternative zur WSDL-Datei ist eine API-Dokumentation. Diese beschreibt in natürlicher Sprache, wie das Interface aufgebaut ist, welche Aufrufsemantik die einzelnen Operationen haben und vieles mehr.

Die kompakteste Darstellung findet man in der WSDL-Datei. Wenn diese aber keine Aufschlüsse über bestimmte Fragen gibt, sollte man sich der API-Dokumentation widmen. Zu guter Letzt gibt es immer noch die Möglichkeit, in Foren zu fragen/suchen oder den Service Administrator zu befragen.

Wollen wir nun am Beispiel des *google Web Service* herausfinden, welche Operationen bereitgestellt werden:

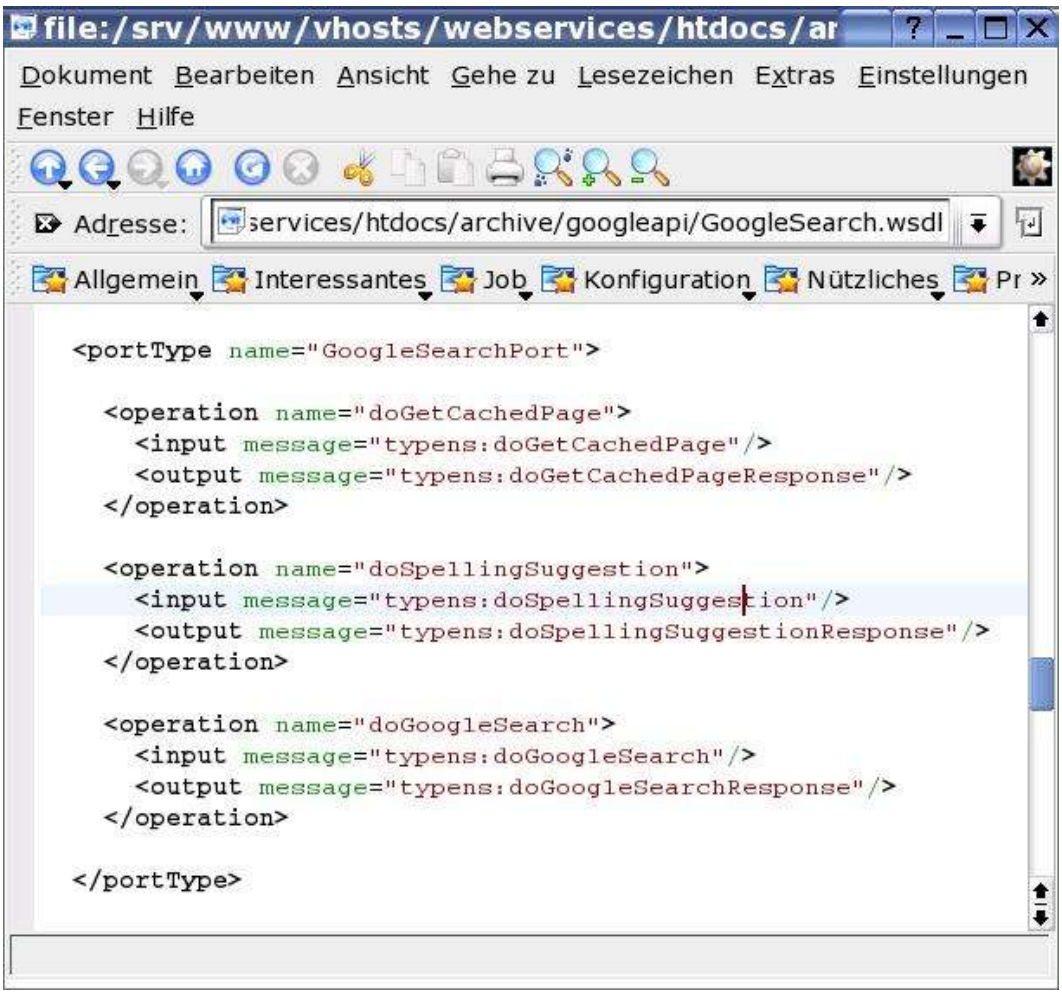

*Abbildung 1: Auszug aus der WSDL-Datei **GoogleSearch.wsdl**: Operationen des Web Service*

In diesem Fall reicht ein kurzer Blick in die Datei *GoogleSearch.wsdl*, die auf der Webseite http://www.google.de/api/ heruntergeladen werden kann. Wie man leicht aus **Abbildung 1** ablesen kann, werden drei Operationen durch den Service bereitgestellt:

- `doGetCachedPageResponse`
- `doSpellingSuggestion`
- `doGoogleSearch`

Anmerkung:
Die Operationen werden im RPC-Style angesprochen, d.h. dass der Aufruf der Operation `doSpellingSuggestion` einem entfernten Prozeduraufruf gleicht, der über die `input`-Nachricht die Parameter erhält und über die `output`-Nachricht das Ergebnis zurückliefert. Die Nachrichten stellen somit die XML-Repräsentation PHP-interner Datentypen dar.

Wir wissen jetzt aber noch nicht, wie man diese Operationen im Detail verwendet. Suchen wir also weiter:

1.2.2. Wie verwendet man diese Operationen?

Auch bei dieser Frage können wir die vorher erwähnten Quellen (WSDL-Datei, API-Dokumentation) verwenden. Betrachten wir die Operation `doGoogleSearch`:

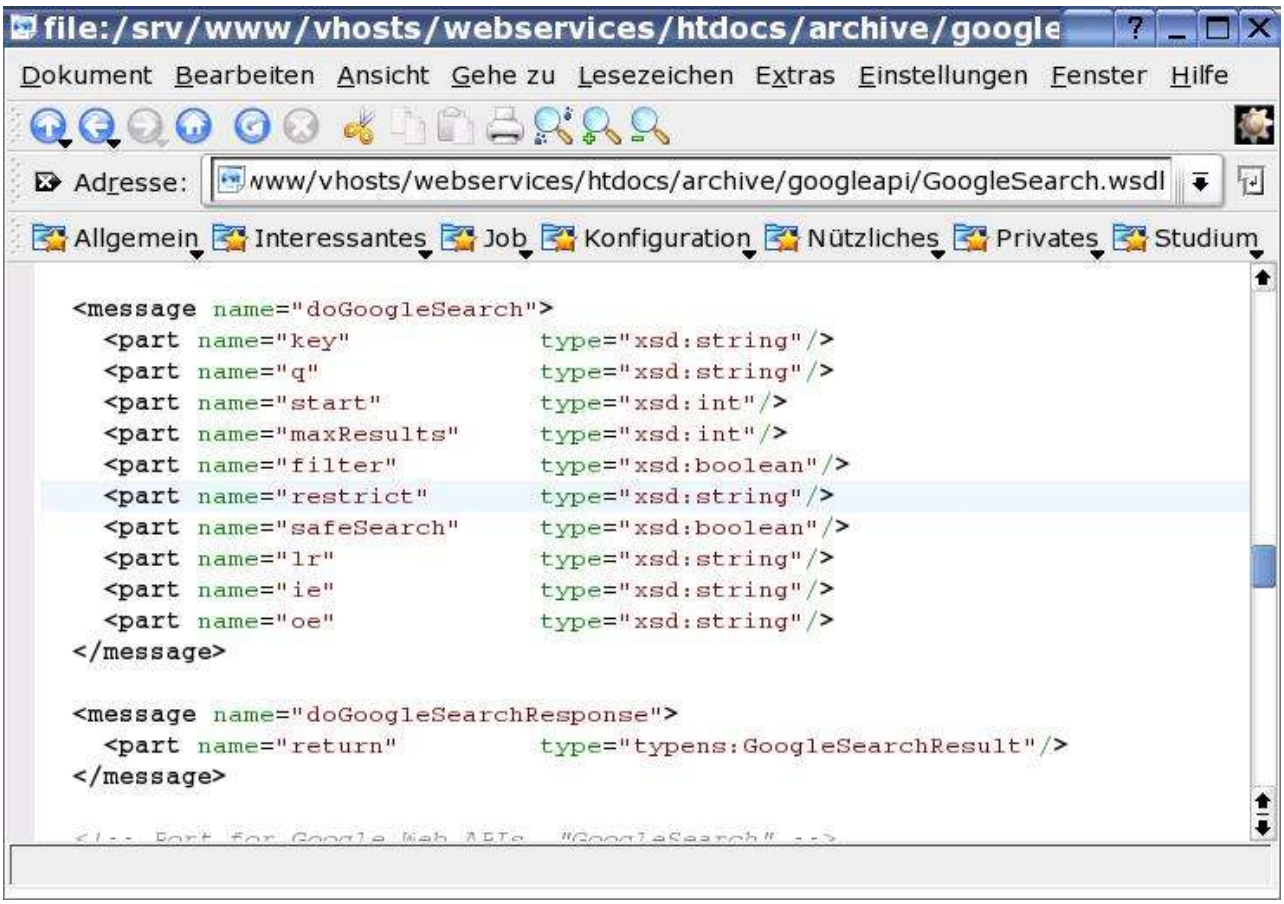

*Abbildung 2: Auszug aus der WSDL-Datei **GoogleSearch.wsdl**: Nachrichten der Operation `doGoogleSearch`*

Die in der obigen **Abbildung 2** gezeigten `message`-Elemente (definiert in der WSDL-Datei) stellen die `input`- und `output`-Nachrichten der Operation `doGoogleSearch` dar. Sie bestehen aus verschiedenen Teilen (`parts`) die sich wiederum aus primitiven (in XML sagt man dazu *simple types*), wie `string`, `int` und `boolean`, oder auch komplexen Datentypen (*complex types*), wie der im eigenen Namensraum definierte `GoogleSearchResult`-Typ, zusammensetzen.

Die Parameter an den Operations-Aufruf von `doGoogleSearch` werden über die Nachricht mit dem Namen `doGoogleSearch` übergeben. Bei Betrachtung des oberen `message`-Elements zeigt sich eine kleine Schwäche der WSDL-Dateien: auf den ersten Blick ist nicht klar, welcher `part` die Zeichenkette für die Suche repräsentiert, und was z.B. der Teil mit dem Namen `oe` bedeutet, wird man anhand der WSDL-Datei wohl nie herausfinden. Deshalb muss man – wahrscheinlich bei den meisten Web Services – noch die API-Dokumentation zu Rate ziehen (denn WSDL-Dateien stellen keine semantischen Beziehungen dar). Dann wird auch klar, dass `q` für den Suchquery steht, `key` den Schlüssel enthalten muss, den man bei der Registrierung bei Google erhält, und z.B. `start` angibt, ab dem wievielten Element Ergebnisse zurückgegeben werden sollen. Wer sich mit den einzelnen `parts` genauer beschäftigen möchte, sei auf die API-Dokumentation (http://www.google.de/api/) verwiesen.

Die Rückgabewerte des Operations-Aufrufs von `doGoogleSearch` werden über die Nachricht mit dem Namen `doGoogleSearchResponse` übergeben. Diese enthält nur einen einzigen Teil vom Typ `GoogleSearchResult`. Da dieser Typ im Namensraum des Services definiert sein muss suchen wir in der WSDL-Datei weiter und finden folgenden Abschnitt:

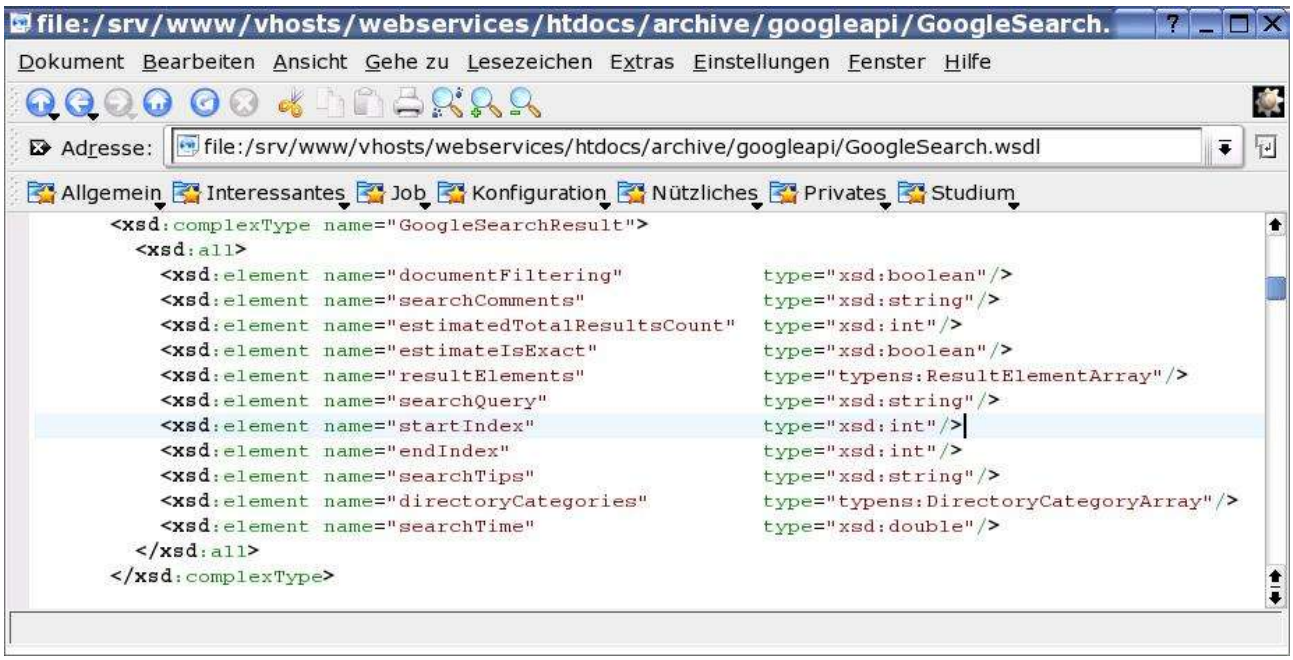

*Abbildung 3: Auszug aus der WSDL-Datei **GoogleSearch.wsdl**: der Typ `GoogleSearchResult`*

In **Abbildung 3** wird die Antwort des Web Services auf Suchanfragen genauer spezifiziert.

Allgemeine Informationen über die Suche werden zurückgegeben, wie die Zeichenkette, nach der gesucht wurde, oder die Zeit, die zum Ausführen der Suche benötigt wurde. Da die Namen der Elemente hier etwas aussagekräftiger gewählt sind als im vorigen Fall, sollte klar sein, was die meisten der Elemente im einzelnen bedeuten. Es gibt aber bestimmt auch Beispiele von Web Services, bei denen nur ein Blick in die API-Dokumentation weiterhilft.

Der für uns interessanteste Teil der obigen Typ-Definition trägt den Namen `resultElements`. Dieses Element soll die auf die Anfrage passenden Dokumente enthalten. Um nun weiter zu forschen, wie ein solches Dokument repräsentiert wird, müssen wir die Definition des Typen `ResultElementArray` suchen:

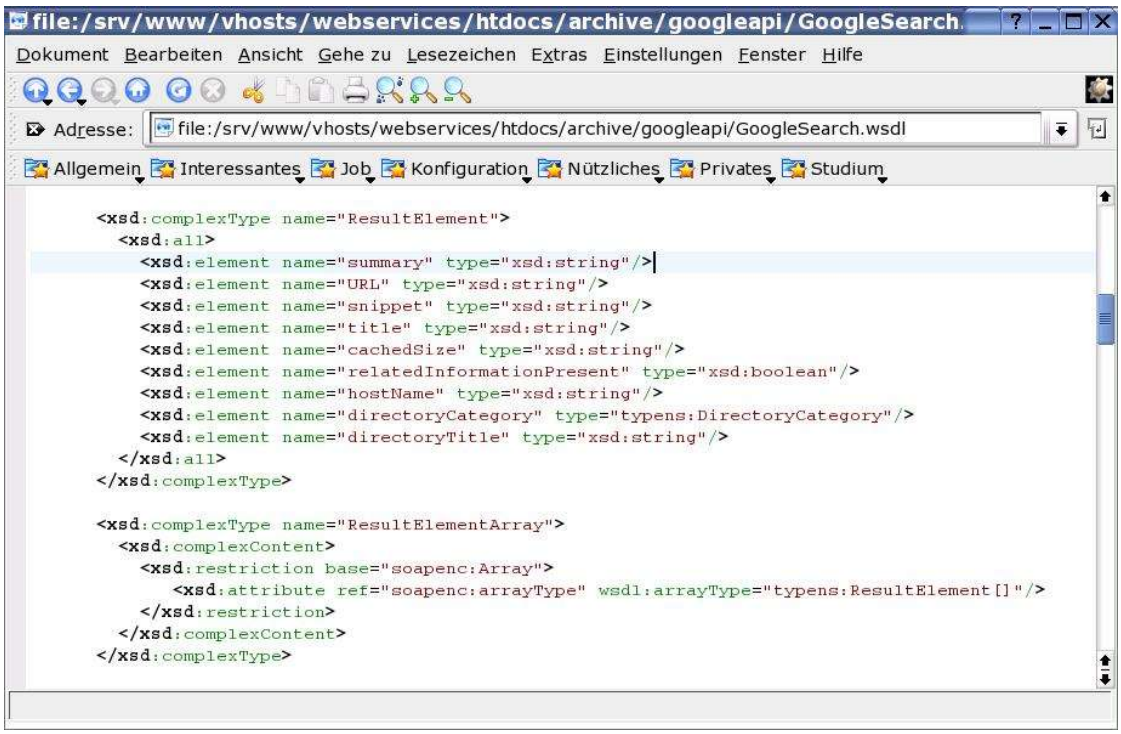

*Abbildung 4: Auszug aus der WSDL-Datei **GoogleSearch.wsdl**: die Typen `ResultElementArray` und `ResultElement`*

Der in **Abbildung 4 (unten)** definierte Typ `ResultElementArray` ist ein spezieller Typ, der vornehmlich in Verbindung mit SOAP-RPC zur Repräsentation von indizierten Feldern (`arrays`) verwendet wird. Felder enthalten gleichartige Elemente, also Elemente vom selben Typ. Diese werden mit ganzen Zahlen beginnend mit `0` durchnummeriert.

Das Ergebnis der *google*-Suche enthält ein Feld mit Elementen vom Typ `ResultElement`. Diese enthalten dann letztendlich die Informationen für diejenigen Dokumente, die auf die Suchanfrage gepasst haben. Dazu gehören Angaben über die URL, unter der das Dokument gefunden werden kann, den Titel des Dokuments und einer kleiner Auszug aus dem Inhalt (`snippet`).

Jetzt haben wir alle wichtigen Informationen zum Aufruf der Schnittstelle *google Web Service*. Wir wissen, welche Anfragen wir auf welche Operation schicken müssen und wie das Ergebnis aussieht. Wie wir aber diese Informationen in einer bestimmten Programmiersprache mit einer bestimmten SOAP-Bibliothek verwenden können, bleibt noch offen. Im folgenden Abschnitt werde ich zeigen, wie man unter PHP die Erweiterung NuSOAP verwendet, um einen Web Service Client zu programmieren.

1.2.3. Wie sende ich Anfragen?

Wir hatten in den Abschnitten zuvor allgemeine Fragen bezüglich der Verwendung von bestehenden Web Services diskutiert. Diese stellen sich egal mit welcher Bibliothek in welcher Programmiersprache man entwickelt.

Betrachten wir folgenden Ausschnitt aus einem Clienten für den *google Web Service*, den ich im Rahmen dieser Seminararbeit zu Demonstrationszwecken programmiert habe:

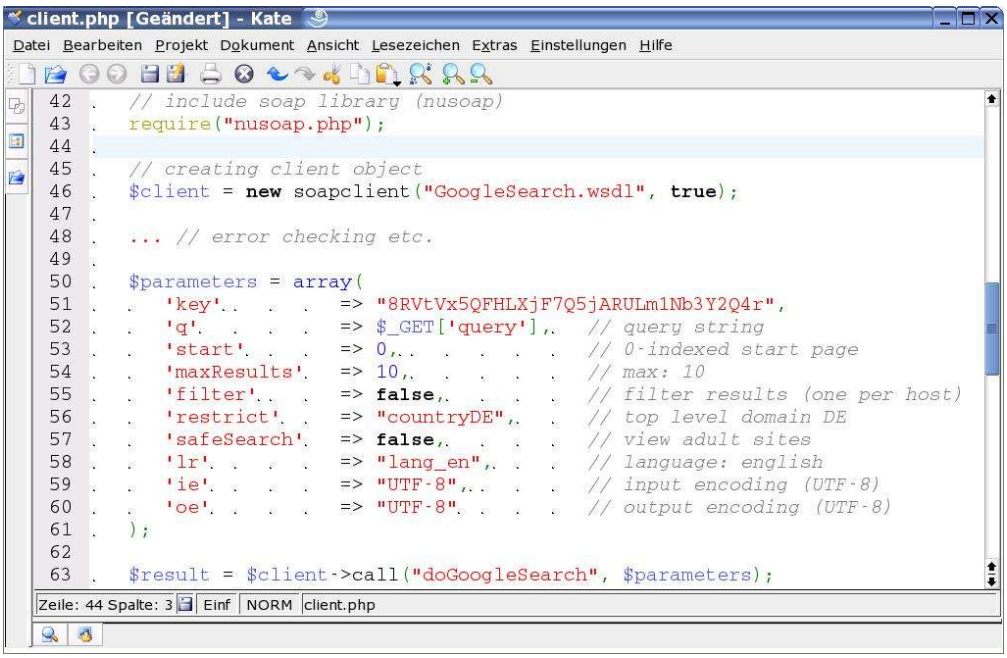

*Abbildung 5: Auszug aus der PHP-Datei **client.php**: Aufruf der Operation `doGoogleSearch`*

Der Code-Ausschnitt in **Abbildung 5** zeigt die Vorbereitung des Aufrufs und den Aufruf der Operation `doGoogleSearch` selbst. Er ist im wesentlichen in drei Teile aufgeteilt:

1. Einbinden der Bibliothek
2. Erstellen eines Clients mit Angabe der WSDL-Datei
3. Aufruf der Operation mit Übergabe der Parameter

Anmerkung:
Die Anweisungen zur Vorbereitung und Initialisierung der Umgebung sollten keiner Erklärung bedürfen. Wer sich damit jedoch näher auseinander setzten will, sollte sich erst einmal in PHP einarbeiten, und dann NuSOAP Tutorials durcharbeiten.

Um nun die Nachricht zum Aufruf der Operation `doGoogleSearch` zu erzeugen, wird ein assoziatives Array namens `parameters` angelegt. An dieser Stelle sei noch einmal auf die **Abbildung 2** verwiesen, in der die Nachricht spezifiziert wurde. In beiden Abbildungen stehen verschiedene Repräsentationen ein und derselben Struktur. Das Konvertieren von jeweils der einen in die andere kann NuSOAP automatisch übernehmen, so aber auch die meisten anderen SOAP-Implementierungen. Dies funktioniert aber nur in Verbindung mit SOAP-RPC und SOAP encoding, wenn man den *document*/*literal*-Stil verwendet, müssen die Nachrichten selbstständig geparst und konvertiert werden, dazu kann man aber natürlich auch bestehende Tools wie *expat* (ein beliebter SAX-Parser) verwenden.

Zu guter Letzt wird nun noch das Ergebnis des Aufrufs über die Objekt-Methode `call(operation[, input])` in die Variable mit dem Namen `result` geschrieben.

1.2.4. Wie werte ich das Ergebnis aus?

Wie auch schon im Falle der Parameter besteht das Ergebnis aus einer PHP-internen Repräsentation der Daten, welche über die SOAP-Nachricht ausgetauscht wurden. Deshalb erfolgt der Zugriff auf die einzelnen Informationen auf die in PHP/NuSOAP übliche weise:

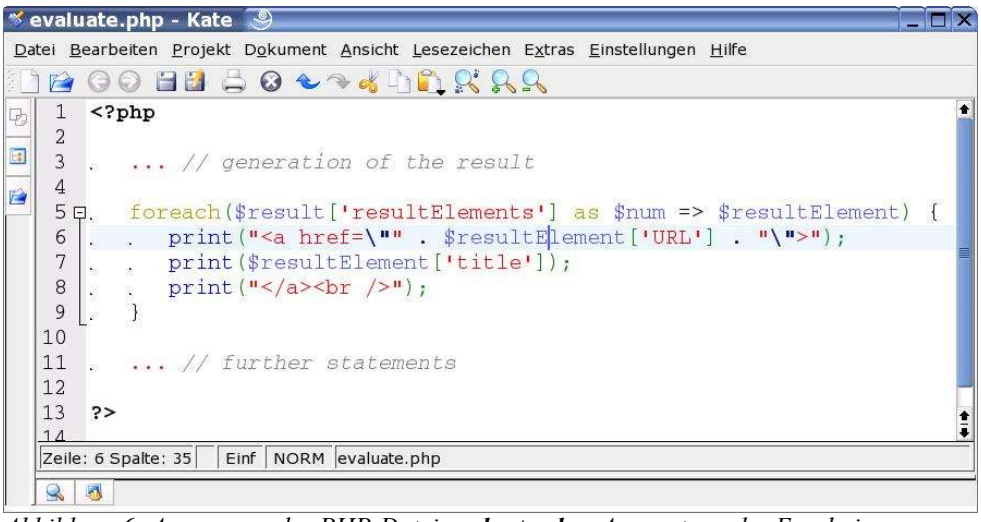

*Abbildung 6: Auszug aus der PHP-Datei **evaluate.php**: Auswertung des Ergebnisses*

Im Code-Fragment aus **Abbildung 6** wird über alle Ergebnis-Dokumente iteriert und jeweils ein Link mit Angabe des Titels ausgegeben. Der Zugriff auf andere Informationen, die in einem `GoogleSearchResult` enthalten sind, erfolgt analog.

Das war im Großen und Ganzen alles, was man wissen muss, um Web Services zu konsumieren.

1.2.5. Der Demo-Client ...

Ich habe natürlich auch eine Version meines Demo-Clients zum testen in das Internet gestellt. Man kann ihn unter der Adresse http://www.fabrizio-branca.de/webservices/google/client.php erreichen.

Unterstützt ist die Suche nach Dokumenten durch Eingabe einer Zeichenkette (wie auf http://www.google.de/). Angezeigt werden die ersten zehn Ergebnis-Dokumente, da der Web Service es nicht erlaubt, mehr Dokumente auf einmal zurückzugeben. Man könnte auch eine Navigation nach Seiten einbauen, da man Informationen über die Gesamtanzahl der auf die Suche passenden Dokumente erhält und der Index des ersten zu liefernden Dokuments angegeben werden kann. Das bleibt jedoch denen überlassen, die planen, den Client im Realfall einzusetzen.

Damit man auch in die internen Vorgänge während der Abfrage von Web Services näher einsehen kann, werden nach jedem Operationsaufruf die SOAP-Nachrichten, die ausgetauscht wurden, mit den entsprechenden HTTP-Headern auf die Festplatte geschrieben (Links am Anfang der Seite verweisen auf diese Dateien). Ausserdem lässt sich eine Art *debug*-Modus aufrufen, der am Ende der Seite die `result`-Variable (assoziatives Array) und *debug*-Informationen der Bibliothek NuSOAP ausgibt.

Damit wäre das Konsumieren von Web Services abgeschlossen. im folgenden Abschnitt soll nun demonstriert werden, wie man einen eigenen Web Service entwickeln und anbieten kann.

1.3. Erstellen von Web Services

Da sich das Prinzip der Fragen im vorigen Abschnitt bewährt hat, wollen wir uns auch am Anfang dieses Abschnitts einmal grundsätzliche Fragen überlegen, welche wir im Zuge der Entwicklung von Web Services beantworten müssen.

Da wäre zum einen natürlich das Problem, was für einen Web Service wir überhaupt anbieten wollen. Damit ist gemeint, ob man eine Schnittstelle zur Buchung von Flügen für ein Reisebüro entwerfen will oder ein Auktionshaus eine Schnittstelle, die dem Nutzer die Interaktion mit dem Auktions-system ermöglicht. Wenn dies feststeht muss eine Infrastruktur durch die Definition von benötigten Operationen geschaffen werden (hier sollte einem Java-Programmierer auch wieder die Parallele zum Entwurf von `interface`-Klassen auffallen). Wichtig dabei ist natürlich wieder einmal die Aufrufsemantik, also die Angabe von `input`- und `output`-Nachrichten. Deshalb sollte man sich auch die Frage stellen, wie man diese Operationen verwenden können soll.

Nach Klärung dieser eher allgemeinen Probleme, muss man sich mit der SOAP-Bibliothek vertraut machen, welche man einsetzten muss. Da wir einen Service entwerfen, müssen wir grob gesagt Anfragen auswerten und Ergebnisse zurückliefern. Die Anfragen kommen als SOAP-Nachrichten, müssen daher erst für die einzelnen Operationen aufbereitet werden, erst dann könne sie verarbeitet werden. Auch das Erzeugen von Ergebnissen erfolgt in unterschiedlichen Schritten, je nach SOAP-Bibliothek.

Wie man diese Probleme angehen kann, soll nun am Beispiel eines Adressbuchs demonstriert werden. Das Adressbuch soll auf einer *mysql*-Datenbank aufsetzten, in der die Einträge gespeichert liegen. Ein Datensatz besteht aus dem Namen, der Telefonnummer und der eMail-Adresse der Person und einer globalen ID zur Identifikation des Datensatzes.

1.3.1. Welche Operationen sollen bereitgestellt werden?

Die Frage kann man in diesem Fall auch so umformulieren: Was soll der Benutzer machen können? Und das ist relativ leicht zu beantworten. Als erstes sollte die Möglichkeit bestehen, alle Datensätze abzufragen. Deshalb müssen wir eine Operation mit dem Namen `getAllAddresses` entwerfen. Weiter soll das Hinzufügen und Löschen von Einträgen gestattet sein (`addAddress`, `removeAddress`) sowie das Durchsuchen der Datenbank anhand einer Zeichenkette mit Volltext-Vergleichen (`search`).

Dinge wie Autorisierung und Authentifizierung wurden hier noch nicht berücksichtigt. Deshalb kann jeder Benutzer, der die URL des Services kennt, sofort alle Daten lesen und schreiben. Im Realfall würde dies wahrscheinlich nicht gewünscht sein, aber zu Zwecken der Demonstration hat dies auch Vorteile.

Wollen wir uns nun am Beispiel der Suche überlegen, welche Parameter man an die Funktion, die den Algorithmus zur Abarbeitung der entsprechenden Operation darstellen, übergeben muss, und welche Daten zurückgeliefert werden sollen.

1.3.2. Welche Parameter benötigen die Funktionen?

Bei der Suche ist die Aufgabe eher trivial. Man will nach einer Zeichenkette suchen und muss deshalb auch eine Zeichenkette übergeben. Zurückgeliefert werden soll die Liste der passenden Adressbucheinträge, die ihrerseits wieder Strukturen darstellen, die sich aus der ID, dem Namen, der Telefonnummer und der eMail-Adresse zusammensetzten (den Daten in der Datenbank).

Wir haben also unsere Definition von `input`- und `output`-Nachrichten gefunden:

- `input`
  Query-Element, das die Zeichenkette enthält, nach der gesucht werden soll
- `output`
  Array-Element, das die `address`-Strukturen enthält, die auf die Suche gepasst haben

Genauso muss man auch für die übrigen Operationen vorgehen, bis die Schnittstelle vollständig definiert ist. Danach kann man anfangen, den Service zu implementieren. Dabei stellen sich aber sofort neue Fragen, sofern man mit der SOAP-Bibliothek noch nicht vertraut ist.

1.3.3. Wie verarbeite ich Anfragen?

Das Verarbeiten von Anfragen erfolgt unter Verwendung von NuSOAP im Wesentlichen in drei Schritten:

1. Hinzufügen der Datentypen
2. Programmieren der Funktion
3. Registrieren der Operation

Bevor man dies alles angehen kann, muss man zuvor noch ein Server-Objekt erstellen, das später die

Konvertierung von Datentypen in SOAP-Nachrichten übernimmt und die Verteilung der einzelnen Nachrichten an die Funktionen, welche den Algorithmus für eine bestimmte Operation darstellen.

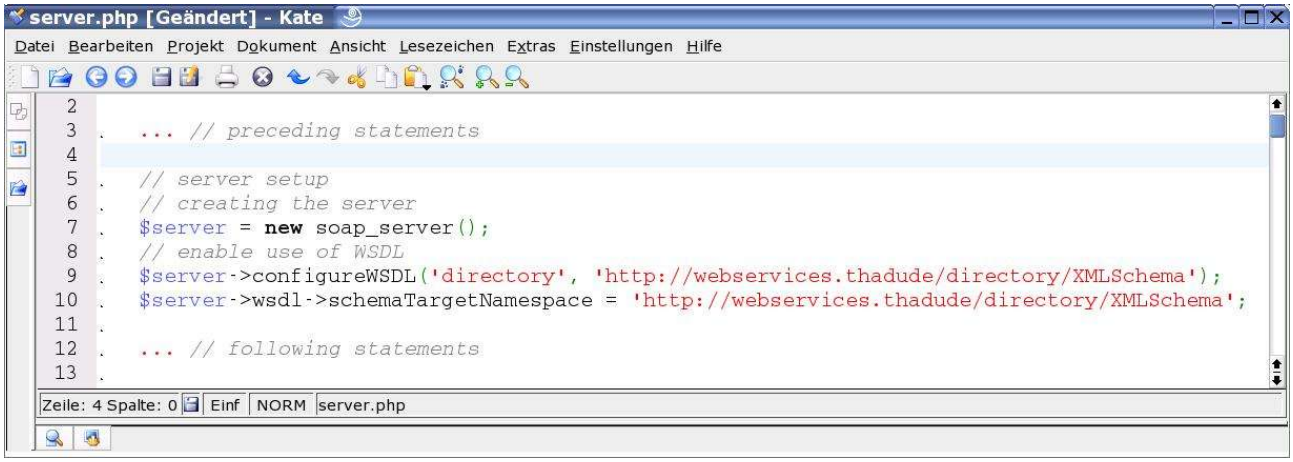

*Abbildung 7: Auszug aus der PHP-Datei **server.php**: Erstellen des Server-Objekts mir WSDL-Unterstützung*

In **Abbildung 7** wird gezeigt, wie man einen Server mit NuSOAP erstellt und diesen für die Verwendung in Verbindung mit WSDL konfiguriert. Jetzt können wir loslegen, das Verarbeiten von Anfragen zu ermöglichen.

1.3.3.1. Hinzufügen der Datentypen

Betrachten wir wieder die `search`-Operation: wie wir bereits vorher festgestellt haben, benötigen wir die Typen `string` (zur Übergabe der Suchanfrage), `addresses` (zur Repräsentation einer Liste von `address`-Datensätzen) und somit auch den Typ `address` (zur Darstellung eines einzigen Adressbucheintrags).

Da Zeichenketten als Typ in XML bereits vorhanden sind, müssen wir nur noch `addresses` und `address` definieren.

In **Abbildung 8** wird demonstriert, wie dies unter Verwendung von NuSOAP gemacht wird. Wichtig dabei ist – egal welche Implementierung man verwendet – ‚dass `addresses` als `array`-Typ verwendet werden kann. Deshalb muss durch Angabe einer `restriction base` der gewünschte Basistyp aus der SOAP encoding Spezifikation integriert werden. Durch Modifikation des Attributs `arrayType` kann dieser Typ dann für `address`-Datensätze genutzt werden. Auffällig in unterer Abbildung ist die Angabe des PHP Typs: `array`. Diese Angabe wird von NuSOAP zur automatischen En- bzw. Decodierung verwendet, d.h. dass eine `addresses`-Struktur in ein indiziertes Feld übersetzt wird.

Ein weiterer möglicher Typ wäre `struct`, der die Repräsentation als assoziatives Array erzwingt. Diesen setzten wir ein zur Definition des `address`-Typs. Somit ist es möglich, die Einzelnen Daten mit dem entsprechenden Schlüssel anzusprechen, z.B. `name`, `email`, ... Wie das aber genau funktioniert, werden wir bei der Implementierung sehen.

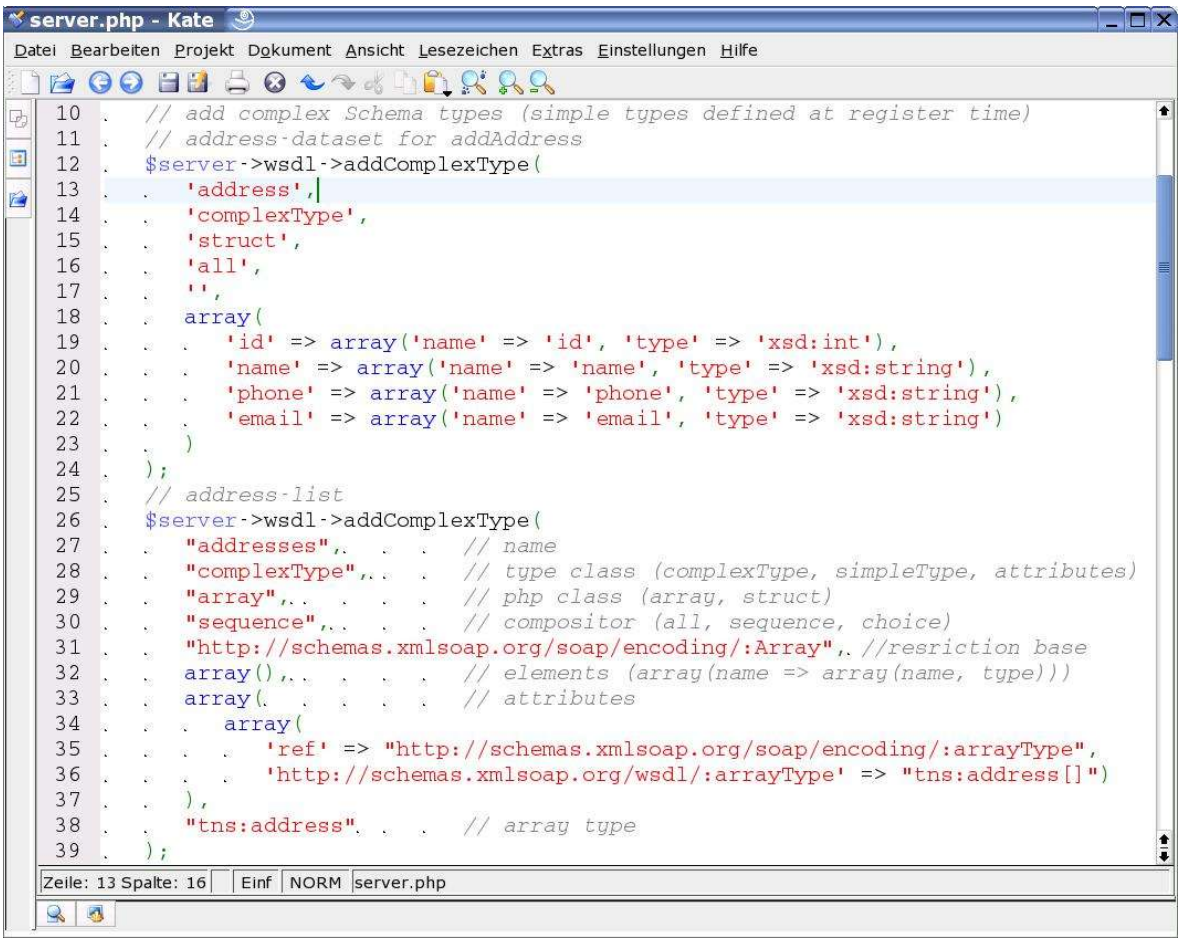

*Abbildung 8: Auszug aus der PHP-Datei **server.php**: Hinzufügen der Datentypen*

Nun haben wir die Datentypen definiert, die zum Austausch von SOAP-Nachrichten für die Operation `search` notwendig sind (für die anderen Operationen geht es analog). Im nächsten Schritt müssen wir die Funktion programmieren, die als Operation veröffentlicht werden soll.

1.3.3.2. Programmieren der Funktion

Erinnern wir uns an die Aufrufsemantik der Operation `search`: Die Funktion erhält einen `string` und soll die Liste der passenden Datensätze zurückgeben.

Die Funktion in PHP hat tatsächlich dieselbe Struktur. Sie bekommt einen Parameter, der die Zeichenkette enthalten soll, und sie liefert ein indiziertes Array mit Elementen, die ihrerseits wieder aus assoziativen Strukturen bestehen – also die PHP-interne Repräsentation der Datentypen `addresses` und `address`.

Da PHP ein *loosley-typed system* ist, können diese Bedingungen leider nicht bei der Deklaration von Funktionen mit angegeben werden, wie es in Java oder C++ der Fall ist. Deshalb muss eine Typ-Überprüfung innerhalb der Funktion vorgenommen werden, um Inkonsistenzen festzustellen. Wenn dabei ein Fehler auftritt oder die Typen wirklich inkonsistent sind, ist es gute Sitte, einen SOAP-Fault zurückzuliefern. Diese können mit der Klasse `soap_fault` erzeugt werden.

Wenn das Ergebnis richtig erzeugt werden konnte, kann man als Rückgabewert der Einfachheit halber das assoziative Array verwenden. NuSOAP encodiert es dann automatisch in eine XML-Repräsentation, die innerhalb der SOAP-Nachricht versandt werden kann.

In **Abbildung 9** ist nun ein Ausschnitt aus dem Code gezeigt, den ich im Rahmen des Beispiel Web Services geschrieben habe. Er zeigt das Gerüst der Funktion `search` mit Typ-Überprüfung, Fehlererzeugung und Ergebniserzeugung.

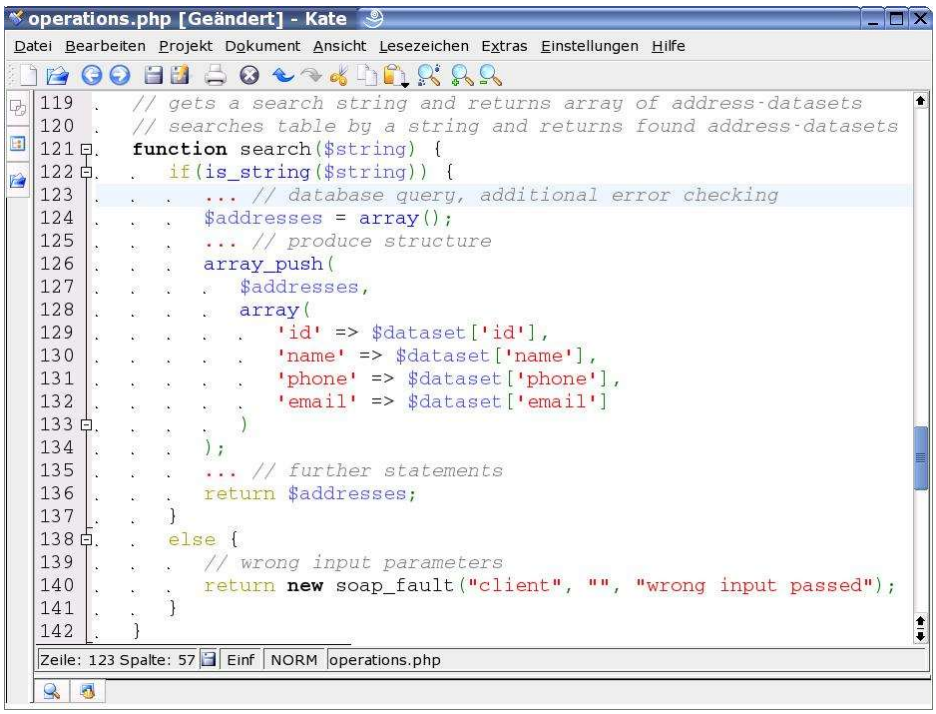

*Abbildung 9: Auszug aus der PHP-Datei **operations.php**: Funktion für die Suche*

Zuletzt muss diese Funktion noch im Server registriert werden, damit sie der Öffentlichkeit zur Verfügung steht. Wie das geht, wird im nächsten Abschnitt gezeigt.

1.3.3.3. Registrieren der Operation

Registrieren bedeutet, dass man angibt, welche Funktionen vom Server/Client aufgerufen werden können, welche Eingabeparameter die Funktionen benötigen und welche Rückgabewerte sie liefern. Wichtig ist dabei vor allem die Verwendung und der Stil der Operation (`style` und `use`). Alle Beispiele, die ich im Rahmen dieser Arbeit verwendet habe, haben mit der Kombination *rpc*/*encoded* gearbeitet. Dies ist in Verbindung mit NuSOAP durchaus zur Zeit noch sinnvoll, da es wesentlich eleganter benutzt werden kann. Wie sich das künftig ändert, kann ich nicht sagen. Sehr viele Quellen haben aber *rpc*/*encoded* als Spezialfall von *document*/*literal* beschrieben und deshalb die zweite Methode empfohlen.

Wollen wir uns das einmal in der Praxis ansehen:

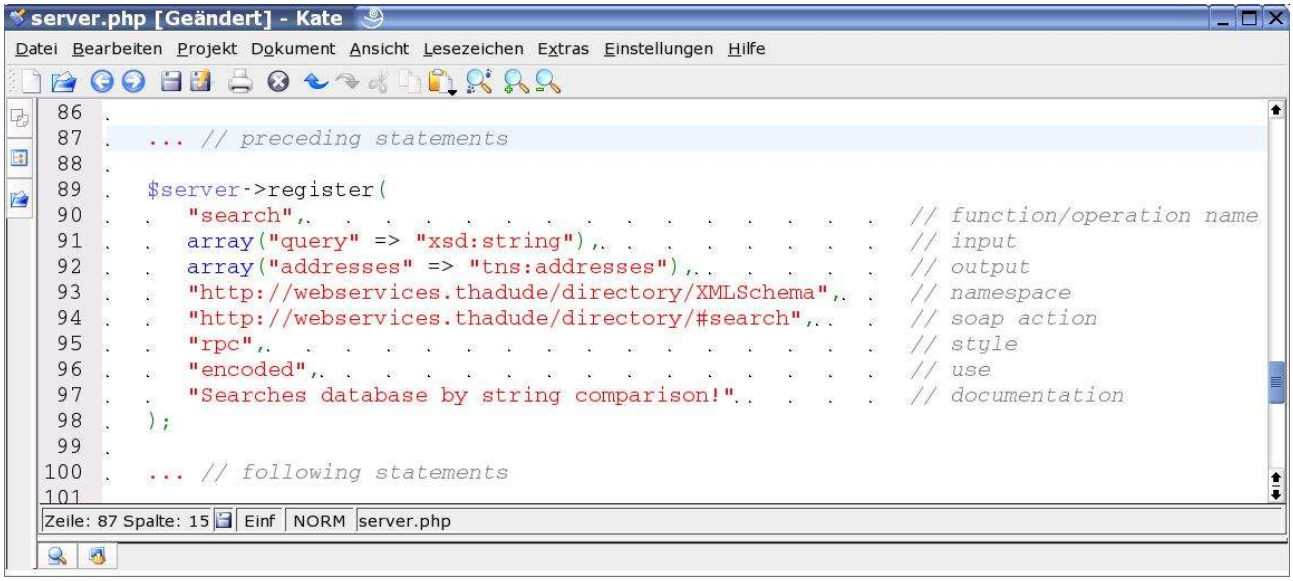

*Abbildung 10: Auszug aus der PHP-Datei **server.php**: Registrierung der Operation `search`*

Das Code-Fragment auf **Abbildung 10** zeigt die Registrierung der Operation `search` unter Verwendung der Bibliothek NuSOAP.

Wichtig ist, dass der Name der Operation mit dem Namen der PHP-Funktion übereinstimmen muss. Es können noch `input`- und `output`-Nachrichten angegeben werden. In obigem Beispiel bekommt die Operation als `input` ein Element namens `query`, dessen Inhalt vom Typ `string` sein soll (also die Suchanfrage). Die Definition des Output erfolgt genauso (Angabe von Elementname und Typ). Man kann optional auch noch einen Dokumentationstext angeben. Dieser wird dann in die WSDL-Datei aufgenommen, die NuSOAP nach Registrierung von Datentypen und Operationen automatisch erstellen kann. Um diese Datei anzuzeigen, muss man nur im Browser (sofern man HTTP als Zugangs-Protokoll verwendet) die Adresse des Dienstes anwählen. Man erhält dann eine dynamisch erzeugte Übersichtsseite, auf der auch ein Verweis auf die WSDL-Datei zu finden ist.

Der Service kann jetzt aber noch nicht als solcher angesprochen werden, d.h. man kann noch keine SOAP-Nachrichten schicken, die er auswertet. Dazu müssen wir den Service sozusagen noch „veröffentlichen“.

1.3.4. Wie veröffentliche ich den Web Service?

Das ist nun wirklich der einfachste und kürzeste Schritt des ganzen Entwicklungsprozesses von Web Services. Das einzige, was nämlich noch getan werden muss nach Erzeugung des Services, ist die Übergabe der über die HTTP-Nachricht erhaltenen POST-Daten an die Methode `service` des Servers.

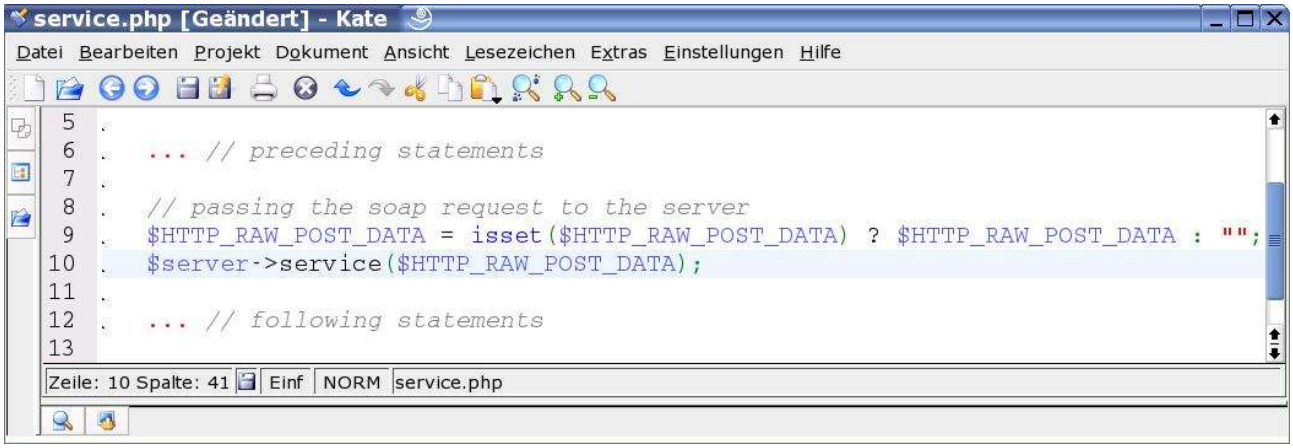

*Abbildung 11: Auszug aus der PHP-Datei **service.php**: Veröffentlichung des Services*

Jetzt ist es möglich, SOAP-Nachrichten an den Service zu schicken, die dann verarbeitet werden können. Ergebnisse liefert das Server-Objekt selbstständig zurück.

1.3.5. Der Demo-Service ...

Den gerade besprochenen Adressbuch Web Service kann man unter der Adresse http://www.fabrizio-branca.de/webservices/service.php finden. Wenn man direkt auf diese Adresse geht, erhält man die von NuSOAP generierte Übersichtsseite (im HTML-Format) mit einem Verweis zur WSDL-Datei.

Ich habe auch einen Demo-Client zur Verwendung des Web Service programmiert (http://www.fabrizio-branca.de/webservices/client.php). Dieser unterstützt das Anzeigen aller Adressen, sowie das Hinzufügen, Löschen von Adressen und das Durchsuchen der gesamten Adress-Datenbank.

Falls jemand an den Nachrichten, die ausgetauscht werden, interessiert ist, findet er auf jeder Client-Seite Verweise zu den aktuell ausgetauschten SOAP-Nachrichten sowie deren HTTP-Headern. Ausserdem kann man eine Art *debug*-Modus aufrufen. Dieser sorgt für das Anzeigen der `result`-Variable (Ergebnis des Operationsaufrufs) und der *debug*-Informationen der NuSOAP Bibliothek.

1.4. Fazit

Das entwickeln von und mit Web Services in PHP ist bezüglich der technischen Details eine sehr einfach Aufgabe. Dies wirkt auf den ersten Blick sehr verlockend. Leider birgt es aber in einigen Anwendungsszenarien Probleme in sich. So ist die Verwaltung und Wartung sowie die Erweiterung und Entwicklung von komplexen Application Infrastructures mit PHP und auch in allen anderen Sprachen nicht wirklich sinnvoll, solange man kein Application Framework verwendet. Das einzige Framework, dass ich im Rahmen dieser Arbeit kennen gelernt habe, heißt Krysalis. Inwieweit Krysalis aber den Ansprüchen von großen Business-Anwendungen gerecht werden kann, lässt sich nicht ohne weiteres feststellen. Über dieses Thema könnte man eine eigene Arbeit schreiben.

Da Web Services sich zur Zeit auch immer mehr in Bereichen durchsetzten, die nichts mit der Integration von Business-Anwendungen gemein haben – wie zum Beispiel der *google Web Service* – rückt PHP immer mehr in den Vordergrund. Für derartige sozusagen von einer komplexen Umgebung losgelösten Anwendungen ist es durchaus sinnvoll eine Bibliothek wie NuSOAP in Verbindung mit PHP einzusetzten, da Wartungsaufgaben, wie Änderungen an der Service-Schnittstelle zu integrieren oder die Präsentationsschicht anzupassen, sich auf einen kleinen Bereich des gesamten Systems beschränken.

In eigener Sache:
Vivian Steller, Student an der Universität Ulm, hat im vergangenen Jahr zusammen mit mir ein XML Framework namens *eecoo* für PHP ins Leben gerufen. Wer sich damit auseinander setzten will, sei auf die Seite http://www.eecoo.de verwiesen. Leider gibt es dort derzeit noch sehr wenige Dokumente bezüglich der Nutzung (Stand August 2004) des Systems. Es wird aber aktiv weiterentwickelt und Dokumentationen zur Entwicklung sind geplant und teilweise schon umgesetzt. Für dieses Framework lassen sich auch problemlos Erweiterungen zur Nutzung von SOAP und somit auch Web Services erstellen. Leider müssen sich die Arbeiten zum jetzigen Stand noch auf andere wichtige infrastrukturelle Dinge konzentrieren.