

Web Service Toolkit für PHP5

Stefan Marr, Falko Menge, Gregor Gabrysiak, Martin Sprengel, Michael Perscheid,

Christoph Hartmann, Andreas Meyer und Sebastian Böttner

Hasso-Plattner-Institut für Softwaresystemtechnik

Zusammenfassung—Dieses Paper beschreibt die Ergebnisse unserer Arbeit an einer Sammlung von Werkzeugen für die automatisierte Erstellung von Web Services unter PHP5. Die besonderen Eigenschaften der Sprache machen die Entwicklung von Meta-Spach-Werkzeugen etwas schwierig. Daher wurde die Reflection API von PHP5 erweitert, um die benötigten Informationen über Sprachkonstrukte zu erhalten und Annotationen in PHP5 zu ermöglichen. Auf dieser Basis wurden zunächst ein leistungsfähiger WSDL-Generator und ein spezieller Code-Generator entwickelt. Um eine sichere Authentifizierung für Nutzer von Web Services zu ermöglichen wurde das Username-Token-Profile aus dem WS-Security-Standard für PHP5 implementiert. Dazu wurde der SOAP-Server um einen Handler-Chain-Mechanismus für SOAP-Intermediaries erweitert. Zusätzlich zu den Werkzeugen für SOAP-basierte Web Services entstand eine Reihe von Werkzeugen zur Entwicklung von RESTful Web Services mit PHP5. Ein Administrationstool stellt eine einheitliche grafische Oberfläche für alle Werkzeuge zur Verfügung, um automatisiert SOAP- und REST-Server für eine bestehende Anwendung zu generieren, wodurch eine plattformübergreifende Kommunikation mit anderen Anwendungen möglich wird.

Stichworte—PHP5, Reflection, Annotations, Web Services, WSDL-Generator, SOAP-Handler-Chains, WS-Security, Username-Token-Profile, REST.

I. EINFÜHRUNG

WEB SERVICES sind eines der wichtigsten IT-Themen der letzten Jahre. Sie bieten interoperable Kommunikation über Plattform- und Programmiersprachengrenzen hinweg und stellen aus diesem Grund für Unternehmen ein vielversprechendes Konzept dar, um bestehende interne und unternehmensübergreifende Anwendungen sinnvoll zu integrieren und so Geschäftsprozesse zu optimieren. Das Grundprinzip dabei ist, dass Anwendungen ihre Funktionalität in Form von Diensten anbieten, die über das Internet von anderen Anwendungen genutzt werden können.

Die wichtigsten Grundlagenstandards für Web Services sind das SOAP-Protokoll [1], die *Web Services Description Language (WSDL)* [2] und *Universal Description, Discovery and Integration (UDDI)* [3]. SOAP ist dabei das Kommunikationsprotokoll für den Zugriff auf die Dienste. Mit WSDL werden vertragsähnliche Beschreibungsdokumente erstellt, die ein Anbieter den Nutzern zur Verfügung stellen kann. UDDI ist der Standard für Namensdienste, mit denen Web Services bekannt gemacht und gesucht werden können.

Zur effektiven und interoperablen Nutzung von SOAP und, mit Abstrichen, von WSDL existieren für alle üblichen Programmiersprachen und Plattformen voll ausgereifte Werkzeuge, die für den Produktionsbetrieb eingesetzt werden können. Für UDDI Namensdienste werden meist kommerzielle Pro-

dukte verwendet, die via Web Service in die Anwendungen integriert werden.

In unserer ersten Ausarbeitung „*Einführung in XML Web Services*“ [4] haben wir SOAP, WSDL und UDDI vorgestellt und gezeigt, wie man die Standards mit PHP5 nutzen kann. PHP5 bietet eine Implementierung des SOAP-Protokolls, die sich nutzen lässt, um interoperable SOAP-Server und -Clients zu entwickeln.

Um eine Klasse in PHP5 als Web Service anzubieten, erstellt man zunächst eine Beschreibung der Schnittstelle mit WSDL. Dabei definiert man unter anderem für alle Datentypen mit Hilfe von XML Schema-Ausdrücken, wie sie für das Versenden in SOAP-Nachrichten serialisiert werden sollen. Eine Dokumentation der Methodensemantik kann manuell aus dem Quelltext der Klasse in die Schnittstellen-Beschreibung übertragen werden. Danach schreibt man ein PHP Script, das mit Hilfe der WSDL-Beschreibung einen SOAP Server startet. Dieser erzeugt dann ein Exemplar der dazugehörigen Klasse und beginnt SOAP-Anfragen zu beantworten.

Das ist natürlich bei weitem nicht so komfortabel, wie man es von .NET oder der Java Enterprise Edition gewohnt ist. Die SOAP-Implementierung ist zwar sehr einfach zu benutzen, jedoch ist die manuelle Erstellung von WSDL-Beschreibungen sehr aufwändig und könnte prinzipiell automatisiert werden. Um also professionell in großen Systemen Web Services anzubieten, benötigt man umfangreiche Werkzeugunterstützung. Typischerweise verwendet man in anderen Programmiersprachen Annotationsmechanismen um Klassen und Methoden einer Anwendung zu markieren, die als Web Service verwendet werden sollen. Mit dem so markierten Quelltext wird eine Werkzeugkette angestoßen, die dann WSDL-Beschreibungen, zusätzlichen Code für Stubs und Skeletons und falls nötig Deployment-Deskriptoren generiert. Damit sind die Web Services sofort bereit für den Produktiveinsatz.

Das Ziel unserer Arbeit war es, diese Lücken in der Werkzeugunterstützung für PHP5 zu schließen. Im Folgenden werden zunächst die entwickelten Werkzeuge einzeln vorgestellt. Dabei wird besonders auf die Architektur und die getroffenen Design-Entscheidungen, die zu der jeweiligen Lösung geführt haben, eingegangen. Danach wird in einer umfangreichen Beispielanwendung gezeigt, wie die Werkzeuge zusammen eingesetzt werden können, um eine Anwendung mit Web Service Schnittstellen auszustatten.

II. EXTENDED REFLECTION API

Zunächst besteht das Problem, an Informationen zu gelangen, die in einer Programmiersprache formuliert sind.

Diese Informationen beschreiben die Funktionalität und den Ablauf eines beliebigen Programms sowie die Schnittstellen von Komponenten. In vielen Sprachumgebungen gibt es zu diesem Zweck sogenannte Reflection APIs, welche es mit der Programmiersprache selbst ermöglichen, an die gewünschten Metainformationen zu gelangen.

Die in PHP5 integrierte Reflection API bietet die grundlegenden Möglichkeiten, um zur Laufzeit Informationen zur Struktur der verfügbaren programmiersprachlichen Konstrukte, insbesondere der verfügbaren Klassen, Erweiterungen und Funktionen, zu erhalten. Darüber hinaus bietet sie ebenso die Möglichkeit auf Exemplare dieser Konstrukte zuzugreifen, um den aktuellen Zustand zur Laufzeit zu ermitteln oder zu modifizieren.

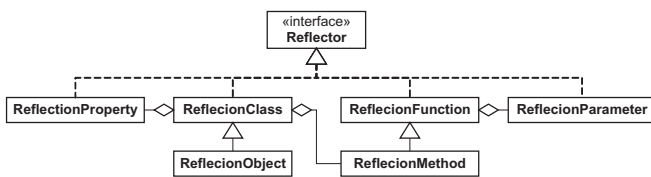


Abb. 1. UML-Darstellung der PHP5 Reflection API

In Abb. 1 ist die grobe Struktur der API dargestellt, welche einem Metamodel für die Definition einer Klasse im Sinne der objektorientierten Programmierung entspricht. Da PHP5 selbst eine schwach typisierte und teils sogar dynamische Sprache ist, bietet die API nur sehr wenige Informationen zur Typisierung der einzelnen Sprachkonstrukte an. Um jedoch nicht nur zur Laufzeit Informationen über die Typisierung erhalten zu können, wurde es erforderlich, die vorhandene API zu erweitern. Ziel dieser Erweiterung ist es zusätzliche Informationen in der Quelltext-Dokumentation der Sprachkonstrukte nutzen zu können. Dabei wurde auf PHPDoc, die in der PHP-Welt genutzte Variation des JavaDoc-Standards, gesetzt. Ursprünglich sind diese Informationen dazu gedacht, die arbeitsteilige Entwicklung zu unterstützen, in dem der Quellcode durch zusätzliche semantische Informationen angereichert wird.

Da die PHPDoc-Tags sich in der PHP-Welt als Standard durchgesetzt haben und in vielen bestehenden Projekten von diesem Gebrauch gemacht wird, bieten sie eine gute Basis um die nötigen Informationen aus den Quelltext-Kommentaren zu extrahieren, ohne neue Verfahren einzuführen, die in bestehenden Anwendungen weitgehende Anpassungen erfordern würden.

Die erweiterte Reflection API nutzt zur Gewinnung der Typinformationen einen einfachen PHPDoc-Parser, welcher die Quellcode-Kommentare untersucht und interpretiert. Um diese Informationen geeignet nutzen zu können, wurde eine programmiersprachliche Repräsentation für ein einfaches Typensystem entworfen.

In Abb. 2 ist das zentrale Interface `Type` dargestellt. Dieses wird im wesentlichen von drei Klassen implementiert. `PrimitiveType` repräsentiert die einfachen Datentypen wie `Integer`, `Float` und `String`, `ArrayType` alle möglichen Array-Typen und `ClassType` alle Klassen. Damit ist es möglich aus einem programmiersprachlichen Konstrukt, nicht nur zu dessen Laufzeit, sondern auch ohne ein Exemplar des

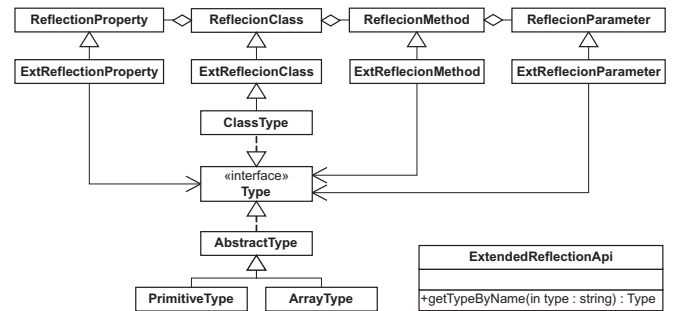


Abb. 2. UML-Darstellung der erweiterten Reflection API

Konstrukts, die gewünschten Typinformationen zu gewinnen.

Dieses Typensystem erlaubt zusätzlich vielfältige eigene Erweiterungen. Durch die Verwendung einer `TypeFactory` ist es möglich für verschiedene Anwendungsaspekte angepasste Typenklassen zu entwickeln, die weitere Zusatzinformationen zur Verfügung stellen. Aktuell implementiert ist unter anderem die Möglichkeit, direkt aus den Typen ihre Definition in XML Schema zu erhalten. Das ist hilfreich beim Erstellen von XML Schema Beschreibungen für Anwendungen, die PHP-Datenstrukturen nach XML serialisieren.

Eine weitere wesentliche Erweiterung ist die Implementierung eines Annotation-Mechanismus, welcher in PHP bis dahin nicht verfügbar war, sich jedoch in anderen gängigen Programmiersprachen zunehmender Beliebtheit erfreut. Für dieses Projekt steht die statische Verwendung von Zusatzinformationen im Vordergrund. So ist es möglich die programmiersprachlichen Konstrukte mit zusätzlichen Eigenschaften zu versehen, unter anderem die Kennzeichnung als Web Service. Zusätzlich denkbar wäre es, diesen Annotation-Mechanismus durch eine geeignete Laufzeitumgebung zu nutzen, um aspektorientierte Programmierung unterstützen zu können. Dies ist in diesem Projekt jedoch nicht realisiert.

III. WSDL GENERATOR

Nachdem die Extended Reflection API das Problem gelöst hatte, Typinformationen für PHP5 Sprachkonstrukte zu ermitteln, konnten wir nun einen WSDL-Generator für die Erstellung von Web Services entwickeln. Die Web Service Description Language (WSDL), welche in [4] näher erläutert wird, verwendet man um die Schnittstellen der Dienste zu beschreiben. Anbieter von Web Services können ihren Nutzern WSDL-Dokumente zur Verfügung stellen, aus denen sie erfahren, welche Operationen angeboten werden und welche Elemente in den SOAP-Nachrichten enthalten sein müssen um sie zu nutzen. Die WSDL-Beschreibung ist dann eine Art Vertrag zwischen den Kommunikationspartnern, an den sich beide halten müssen um erfolgreich Nachrichten auszutauschen. Der von uns entwickelte WSDL-Generator erzeugt WSDL-Dokumente gemäß dem WSDL 1.1 Standard [2].

Bereits bei der Konzeption eines Web Services gibt es zwei grundlegende Entscheidungen zu treffen: die Art des Methodenaufrufs und die Art der Datenrepräsentation in XML. Zuerst widmen wir uns den Aufrufsmodi.

Ein Web Service kann durch zwei verschiedene Arten angestoßen werden - entweder über einen „Remote-Procedure-

Call“ (RPC) oder auf Basis des „Document“-Stils (DOC). Bei RPC wird versucht, durch die Methode und deren Parameter einen Methodenaufrufstack direkt beim Server nachzuahmen. Die so aufgebaute Struktur, die diesen Stack repräsentiert, wird dann zur Kommunikation genutzt. Der Client nimmt den Web Service als einzelne, logische Komponente mit gekapselten Objekten wahr. Die Komplexität wird also durch den SOAP RPC Stack verborgen.

Bei der DOC-Variante hingegen wird der Methodenaufruf direkt auf ein XML Dokument abgebildet, das dann über SOAP versendet wird. Der gesamte Nachrichtenaustausch zwischen Client und Web Service ist hier durch die zugrunde liegende XML Schema Beschreibung bereits vorgegeben. Da die Nachricht die Art und Reihenfolge der Kommunikation bereits vorgibt, muss diese demzufolge nur benannt werden um die Parameterliste direkt übergeben zu können. Dadurch entsteht bei der Kommunikation viel weniger Traffic, als es bei RPC der Fall ist. Während nämlich die Methodenaufrufe durch den Aufbau des Stacks erheblich mehr Overhead erzeugen, muss man bei DOC nur die Nachricht aus dem Schema identifizieren und kann die (benannten) Parameter direkt übergeben. Eine genauere Darstellung der Auswirkungen der verschiedenen SOAP Varianten auf die Performance ist unter [6] zu finden.

Des Weiteren gibt es noch zwei verschiedene Kodierungsmöglichkeiten, um die Übergabewerte beim Sender zu serialisieren und beim Empfänger wieder zu rekonstruieren. Bei der ersten Variante wird eine spezielle Kodierung verwendet. Die einzige gebräuchliche Kodierung ist das im SOAP-Standard [1] spezifizierte SOAP-Encoding. Diese schreibt vor, wie man alle Typen der jeweiligen Programmiersprache, auf die vom Standard definierten Typen von XML-Elementen abbildet. Wenn die Nachricht dann also versendet werden soll, müssen Client und Web Service dafür sorgen, diese Abbildungen zu nutzen. Die zweite Kodierungsart ist die „Literal“-Variante. Hierbei wird für die Nachrichten keine spezielle Kodierung verwendet sondern reines XML, welches über XML Schema zu definieren ist. Der Nachteil hierbei ist, dass alle Datenkonstrukte explizit durch eigene Schemas beschrieben werden müssen.

Diese zwei mal zwei Möglichkeiten lassen sich nun kombinieren und so ergeben sich theoretisch vier Varianten, einen Web Service zu erstellen:

- RPC/Encoded
- RPC/Literal
- Document/Encoded
- Document/Literal

Document/Encoded wird jedoch nicht genutzt, da es sich schon in seiner Konzeption selbst widerspricht. Die Frage bleibt also: welche Kombination ist zu bevorzugen? Die Antwort liefert die Web Services Interoperability Organization (WS-I), ein Industriekonsortium, das sich mit der plattformübergreifenden Kommunikation zwischen Web Services beschäftigt. Web Services, welche die Anforderungen in den WS-I Profilen erfüllen, können interoperabel über Plattform- und Programmiersprachengrenzen hinweg kommunizieren. Das ist nur mit solchen Web Services möglich, die auf den Kombinationen RPC/Literal oder Document/Literal aufbauen. Die Verwendung des SOAP-Encodings oder anderen Kodierungen

wird generell nicht mehr empfohlen. Daher wurde bei der Entwicklung des WSDL-Generators der Fokus in erster Linie auf die Kodierungsart Literal gesetzt. Trotzdem ist er in der Lage, neben den WS-I-konformen Varianten auch das dennoch gebräuchliche RPC/Encoded durch die Erzeugung entsprechender WSDL-Dateien zu unterstützen.

Abb. 3 zeigt die Schnittstelle über die der WSDL-Generator genutzt werden kann. Der Generator selbst wird initialisiert mit der Angabe des Namens für den zu erstellenden Service, dem URI über den der Service erreichbar sein soll, dem Namespace für die WSDL-Beschreibung und der gewünschten SOAP-Bindung. Standardmäßig wird vom Generator die „Document/Literal-Wrapped“-Bindung erzeugt.

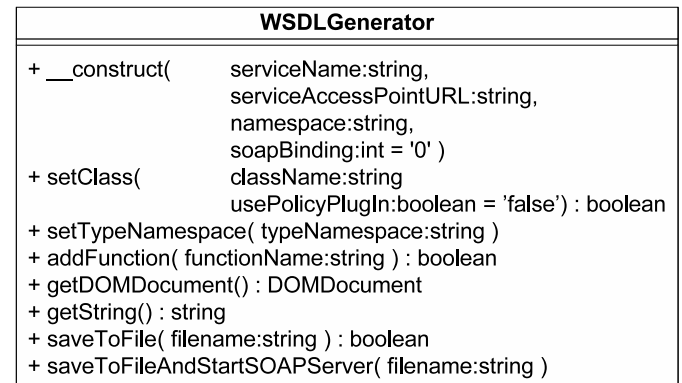


Abb. 3. Klassendiagramm des WSDL-Generators

Der Entwickler kann dann entweder per `setClass()` eine Klasse, deren Funktionalität angeboten werden soll, einlesen lassen oder per `addFunction()` eine beliebige Menge von Methoden zur Erbringung dieser Funktionalität nutzen. Es ist jedoch nicht möglich, diese beiden Methoden der Erstellung zu mischen, da sie sich gegenseitig ausschließen. Ab diesem Punkt wird die Extended Reflection API aktiv. Sie liefert nicht nur die Methodennamen, sondern Listen von `ExtReflectionMethod`-Objekten, an denen direkt Informationen zu Parametern und Rückgabewerten abgefragt werden können. Dies ist ein großer Vorteil für den `WSDLGenerator`, da er so nicht selbst die Klassen parsen muss, um die benötigten Informationen zu erhalten.

Alle so gefundenen Methoden bzw. jede hinzugefügte Funktion muss in die WSDL-Datei abgebildet werden. Sollten bestimmte Methoden nicht veröffentlicht werden, so kann das `Policy-PlugIn`, das im Abschnitt VII-B vorgestellt wird, genutzt werden. Es erhält als Eingabe die Liste der `ExtReflectionMethod`-Objekte und liefert eine gefilterte Liste zurück. Die Information, ob eine Methode veröffentlicht werden soll oder nicht, findet es in der angeschlossenen Datenbank, die der Entwickler zuvor mit Hilfe des `AdminTools` für die entsprechende Klasse füllen muss. Im `AdminTool` kann außerdem eine Beschreibung der Methode in die Datenbank eingepflegt werden, die dann automatisch, sofern das `Policy-PlugIn` genutzt wird, mit in der WSDL-Datei ausgegeben wird.

Der schwierigste Teil der Abbildung von Methoden auf die WSDL-Beschreibung ist die korrekte Definition der Kodierung von komplexen Datentypen, da diese auch auf Clientseite er-

zeugt und verarbeitet werden müssen. Um das zu ermöglichen ist der WSDL-Generator in der Lage, rekursiv XML-Schemas zu erstellen, die alle gebräuchlichen Datentypen abdecken. Per Extended Reflection API werden so z.B. Klassen als solche erkannt. Damit ist der Generator in der Lage, die Klassestruktur rekursiv zu analysieren und das XML Schema sukzessive zu ergänzen, sollten in einer Klasse selbst neue, komplexe Datentypen als Attribute enthalten sein.

Damit dabei kein redundanter Code erzeugt wird, vermerkt der Generator die Namen der erzeugten Typen in einer Liste und greift während der Verarbeitung auch auf diese zu.

Der Generator erzeugt ein DOM-Dokument, das die WSDL-Struktur enthält. Der Aufbau dieser Struktur beginnt mit dem Definition-Element, in dem alle verwendeten Namespaces definiert werden und das alle anderen Beschreibungselemente enthält. Dann folgt die Types-Sektion, die XML Schema Beschreibungen für alle komplexen Datentypen enthält und, bei Verwendung einer Literal-Kombination, außerdem die Anfrage- und Antwortnachrichten definiert. Eine Liste der Methoden ist indirekt durch den Message-Teil gegeben, bei dem für jede Methode ein Element für die Anfrage und ein Element für die entsprechende Antwort vorhanden ist. Darauf folgt die PortType-Abteilung, in der die Methoden auf Operationen abgebildet werden. Anschließend wird die Kombination aus Aufruf und Kodierung explizit in der Binding-Sektion angegeben und abschließend folgt das Service-Element, bei dem der Service an einen Port gebunden wird.

Auf die erzeugte Struktur kann der Anwender dann mit den Methoden `getString()` oder `getDOMDocument()` zugreifen, bzw. sie mit `saveToFile(filename)` in das Dateisystem speichern. Zusätzlich gibt es die Möglichkeit durch `saveToFileAndStartSOAPServer(filename)` direkt einen SOAP-Server zu starten um den Web Service sofort zu testen.

IV. DOCUMENT/WRAPPED-ADAPTER-GENERATOR

Um entfernte Funktionsaufrufe über Web Services mit einer Document/Literal SOAP-Bindung realisieren zu können, wird typischerweise ein Mechanismus verwendet, der auch als Document/Wrapped Bindung bezeichnet wird. Dabei sind in den SOAP-Nachrichten die serialisierten Argumente und Rückgabewerte einer Operation in einem zusätzlichen XML-Element verpackt, so dass im Body der SOAP-Nachricht immer nur ein einzelnes XML-Element mit einem oder mehreren Kind-Elementen vorkommt. Die SOAP-Implementierung von PHP5 wurde entwickelt um alle Arten der SOAP-Bindung zu unterstützen. Diese generische Auslegung führt allerdings dazu, dass das Wrapping und Unwrapping bei Document/Wrapped Web Services nicht automatisch vom SOAP-Server vorgenommen wird, sondern durch die gerufene Methode oder Funktion ausgeführt werden muss. Auf der einen Seite bietet dieses Vorgehen für Anwendungen einige Freiheiten in der Verwendung von SOAP. Andererseits erfordert es, die Signaturen der Methoden oder Funktionen anzupassen, um eine Document/Wrapped Bindung zu nutzen. Eine solche Anpassung verhindert allerdings die parallele Nutzung als

reguläre Methode und ist in bereits bestehenden Systemen völlig undenkbar. Zudem müssten die PHPDoc-Kommentare, um eine WSDL-Generierung zu ermöglichen, die tatsächlichen Typen für Argumente und Rückgabewert verleugnen.

In manchen Klassen findet man beispielsweise folgende Konstruktionen um das Problem zu umgehen:

```
public function echoString($inputString) {
    if (isset($inputString->inputString)) {
        return array(
            'echoStringResult'
            => $inputString->inputString
        );
    } else {
        return $inputString;
    }
}
```

Die Fallunterscheidung macht die Methode wieder in ihrem gewohnten Kontext verwendbar, ist aber nur eine partielle Lösung des Problems, da dies nur für Methoden mit einem einzigen Parameter funktioniert. Allgemein lässt sich das Problem lösen, indem man eine Adapter-Klasse einsetzt, die Methoden mit den gleichen Namen anbietet, und diese anstelle der Original-Klasse beim SOAP-Server registriert. Die Methoden der Adapter-Klasse rufen die Methode an der Original-Klasse mit den Argumenten aus dem vom SOAP-Server übergebenen Objekt. Der jeweilige Rückgabewert wird in ein Array verpackt bevor er zur Serialisierung an den SOAP-Server zurückgegeben wird.

Eine einfache Beispielklasse könnte so aussehen:

```
/**
 * @webservice
 */
class Adder {
    /**
     * @webmethod
     * @param integer $a
     * @param integer $b
     * @return integer
     */
    public function add($a, $b) {
        return $a + $b;
    }
}
```

Für diese Klasse könnte dann ein möglicher Adapter folgendermaßen implementiert werden:

```
class AdderDocumentWrappedAdapter {

    /**
     * @var Adder
     */
    private $target;

    /**
     * @param Adder $target
     */
    public function __construct($target = null) {
        if (empty($target)) {
            $this->target = new Adder();
        } else {
            $this->target = $target;
        }
    }
}
```

```

/**
 * @param object $args
 * @return array<string,integer>
 */
public function add($args) {
    return array(
        'addResult' =>
            $this->target->add($args->a, $args->b)
    );
}
}

```

Die Adapter-Klasse sorgt transparent für das Unwrapping von Argumenten und das Wrapping von Rückgabewerten. Die Original-Klasse muss dazu in keiner Weise angepasst werden.

Ein solcher Adapter enthält keine anwendungsspezifische Geschäftslogik und kann daher automatisch generiert werden. Auf Basis unserer Extended Reflection API haben wir einen entsprechenden Code-Generator entwickelt. Abb. 4 zeigt die Schnittstelle, über die das Werkzeug genutzt werden kann.

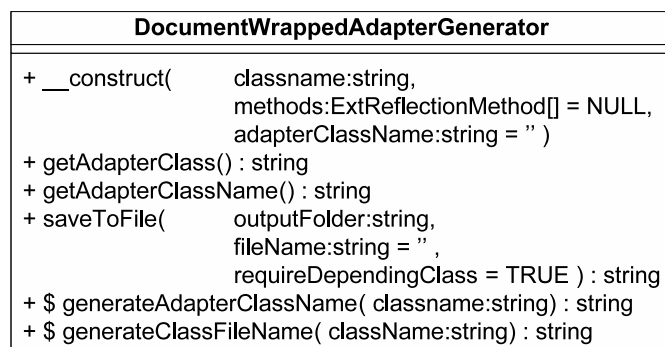


Abb. 4. Klassendiagramm des Document/Wrapped-Adapter-Generators

Zusätzlich zu der Generierung von WSDL-Beschreibungen wird somit auch dieser Schritt für die Erstellung von Web Services mit PHP5 automatisiert.

V. WS-SECURITY

Nicht immer möchte man seine Web Services jedem frei verfügbar machen. Möglicherweise möchte man kostenpflichtige Dienste anbieten oder sie nur unternehmensintern bzw. mit ausgewählten Geschäftspartnern nutzen. Dabei werden die Grenzen der Grundlagenstandards SOAP, WSDL und UDDI schnell erreicht. Dies bedeutet nicht, dass keine möglichen Lösungen existieren. Doch wenn keine einheitlichen Standards verwendet werden, gehen die Lösungen schnell soweit auseinander, dass es an Interoperabilität mangelt und die Entwickler von Client-Applikationen Integrationsprobleme haben. Dieses Kapitel wird sich mit Sicherheitsmechanismen für Web Services auseinander setzen.

A. Anforderungen an gesicherte Web Services

Wird allgemein von Sicherheit gesprochen, sind viele Dinge zu beachten. Dabei handelt es sich meist nicht nur um funktionale, sondern auch um nicht-funktionale Anforderungen. Im Allgemeinen sind dabei die Punkte Datenintegrität, Vertraulichkeit, Authentifikation und Autorisation anerkannt.

Für einige dieser Sicherheitsanforderungen existieren bereits Lösungen, die auch mit PHP genutzt werden können. Die Integrität der Daten wird bereits sichergestellt durch den Einsatz des TCP-Protokolls zur Datenübertragung. Vertraulichkeit kann realisiert werden, indem eine SSL-Verschlüsselung zum Beispiel in Form des HTTPS-Protokolls genutzt wird. Grundvoraussetzung für eine Autorisation ist ein authentifizierter Benutzer. Die Authentifizierung von Benutzern ist der Aspekt mit dem sich Web Services Security (WSS) beschäftigt. Autorisation hingegen ist immer anwendungsspezifisch und kann daher nicht durch Standards abgedeckt werden. Die entscheidende Komponente, die für sichere Web Services mit PHP5 noch fehlte, war also eine Implementierung des WS-Security-Standards.

Für die Umsetzung von Sicherheitsaspekten für Web Services gilt es einige Designziele einzuhalten. Hierzu zählt vor allem, dass die Benutzung von Sicherheitsmechanismen nicht zu Lasten des Anwendungsentwicklers gehen darf. Eine Sicherheits-Implementierung sollte daher nur eine Erweiterung zur Geschäftslogik sein, die bei Bedarf ausgetauscht werden kann. Ein weiteres wichtiges Ziel ist Interoperabilität, denn Web Services dienen zur plattformübergreifenden Kommunikation. Die Sicherheitsmechanismen müssen also beispielsweise problemlos mit .NET oder Java genutzt werden können.

B. Mögliche Ansätze zur Absicherung von Web Services

Ein grober Ablauf für eine mögliche Lösung könnte folgendermaßen aussehen:

- 1) Anmelden, sowie Verschlüsseln der Anmeldedaten auf Seite des Clients
- 2) Die Anmeldedaten auf Serverseite abfragen und auswerten
- 3) Nach der korrekten Authentifizierung wird die Anfrage an die Web Service Implementierung weitergeleitet

Lösungen eines selbst implementierten Token-Prinzips sind immer möglich, stehen jedoch im Kontrast zu einer geforderten Interoperabilität. Demnach muss ein Standard gefunden werden, welcher genau diese Anforderungen erfüllen kann. In der Welt von Java und .NET hat dabei Web Services Security¹ von OASIS [7] eine weite Verbreitung gefunden. Dieser Standard umfasst mehrere Teile. Das ist zum einen SOAP Message Security zum Verschlüsseln von Nachrichten und zur Behandlung von Fehlern. Zum anderen gibt es verschiedene Token Profiles, welche Authentifizierungsmechanismen für Web Services spezifizieren. Wir haben uns damit beschäftigt das Username Token Profile 1.0 für PHP5 umzusetzen.

C. Username Token Profile 1.0 im Detail

Das Username Token Profile [8] wurde von OASIS erarbeitet. Unsere Umsetzung basiert auf Version 1.0 dieses Standards. Die Version 1.1 lag zu Beginn unserer Arbeit nur als Entwurf vor und der erweiterte Funktionsumfang wurde nicht benötigt. Des Weiteren wird Version 1.0 bereits durch diverse Middleware-Plattformen unterstützt und verspricht somit eine gute Interoperabilität. Der Standard beschreibt keine

¹Basiert auf Standard 1.0 (am 17.2.2006 wurde Version 1.1 veröffentlicht)

neue Techniken, sondern gibt einen Rahmen vor, wie die benötigten Informationen für eine Authentifizierung in einem standardisierten Format zu übertragen sind. Diese Informationen umfassen zum Beispiel Benutzername und Passwort, aber auch technische Details wie Angaben zum verwendeten Verschlüsselungsverfahren.

```
<wsse:Security>
  <wsse:UsernameToken>
    <wsse:Username>Micha</wsse:Username>
    <wsse:Password Type="..#PasswordDigest">
      weYI3nXd8LjMNVksCKFV8t3rgHh3Rw==
    </wsse:Password>
    <wsse:Nonce>WScqanjCEAC4mQoBE07sAQ==</wsse:Nonce>
    <wsu:Created>2006-02-16T01:24:32Z</wsu:Created>
  </wsse:UsernameToken>
</wsse:Security>
```

Abb. 5. Beispiel für die Elemente des Username Token Profils im SOAP-Header

Das Format definiert ein neues Element `<wsse:Security>` im SOAP-Header, welches weitere Elemente mit Web Services Security Daten kapselt. Darauf folgt ein Element `<wsse:UsernameToken>`, das die Verwendung des Username Token Profils signalisiert und die benötigten Daten-Elemente enthält. Das Datenelement `<wsse:Username>` muss immer angegeben werden und enthält den unverschlüsselten Benutzernamen. Das nächste Element `<wsse:Password Type="..#">` enthält das Passwort des Benutzers oder Passwortäquivalent. Beim Attribut `Type` dieses Elements gibt es zwei mögliche Werte:

- *PasswordText (Standardwert)* Hierbei kann das `Type`-Feld leer gelassen werden und als Daten sind nur der Username und das Passwort (bzw. Passwortäquivalent) notwendig.
- *PasswordDigest* Beim `Type`-Wert `PasswordDigest` wird das Passwort verschlüsselt übertragen. Dabei werden vom Standard keine Verschlüsselungsalgorithmen festgelegt, da diese austauschbar sein sollen. Um sicherzustellen, dass keine Anfrage zweimal gesendet werden kann (Replay-Attacke), wird das Passwort (`Password`) zusammen mit einem Zeitstempel (`Created`) und einer Zufallszahl (`Nonce`) verschlüsselt. Die Formel, nach der sich die Verschlüsselung zusammensetzt, sieht nun wie folgt aus:

$$\text{PasswordDigest} = \text{Base64}(\text{SHA-1}(\text{Nonce} + \text{Created} + \text{Password}))$$

Passwort, Zeitstempel und Zufallszahl sind Zeichenketten, die aneinander gehängt, mit dem SHA-1-Algorithmus verschlüsselt und Base64 kodiert werden. Zufallszahl und Zeitstempel werden ebenfalls im SOAP-Header mitgeschickt. Der Server kann mit diesen beiden Informationen und dem ihm bekannten Passwort einen Vergleichswert errechnen, um damit die Korrektheit der Anfrage zu überprüfen. Somit ist jede Anfrage nur genau einmal gültig, denn das Ergebnis der Formel ist jedes Mal verschieden. Identische Anfragen werden vom Server abgewiesen.

Die letzten beiden Elemente werden nur für den Passtworttyp `PasswordDigest` benötigt und senden zum einen die Zufallszahl (`<wsse:Nonce>`) im Base64-Format und zum anderen den Erzeugungszeitpunkt (`<wsu:Created>`) mit.

Bei den Elementen werden die XML Namensräume `wsse` (<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd>) und `wsu` (<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd>) verwendet.

Der Standard sieht vor, dass Anfragen mit einem alten Zeitstempel zurückgewiesen werden und ein Cache mit schon benutzten Zufallszahlen geführt wird. Als Minimum gibt der Standard eine Empfehlung von fünf Minuten an.

Passwortäquivalent bedeutet in diesem Zusammenhang, dass dieses extern verschlüsselt sein kann und sollte. Bei unserer Umsetzung gilt grundsätzlich das Passwörter immer MD5-verschlüsselt sein müssen. Dementsprechend gilt auch beim `PasswordText`, dass ein Passwort einem MD5(`Password`) entspricht und nicht nur durch ein reines Passwort repräsentiert wird.

Für weitere Details sei auf den Standard [8] verwiesen. Es wurden an dieser Stelle kurz und knapp die grundlegenden Ideen des Standards zu vermittelt. Im Nachfolgenden soll die Umsetzung in PHP näher beleuchtet werden.

D. Erweiterter SOAP-Server mit WS-Security

1) *Vorhandene Komponenten:* PHP bietet seit der Version 5 eine Implementierung des SOAP-Protokolls in Form eines Erweiterungsmoduls. Mit dem Modul lassen sich eigene Web Services erstellen und vorhandene Dienste nutzen. Sicherheitsmechanismen sind darin allerdings noch nicht integriert. Die Umsetzung des Username Token Profiles für PHP wird auf Basis der vorhandenen Klassen `SoapClient` und `SoapServer` erfolgen. Die erstellten PHP Klassen erben von diesen bzw. erweitern sie um die zusätzlich benötigten Aspekte. Dies hat den Vorteil, dass das Abarbeiten von SOAP-Nachrichten nicht nochmal neu implementiert werden muss. Im Idealfall profitiert die erstellte Lösung auch von zukünftigen Verbesserungen des SOAP-Erweiterungsmoduls. Das Erweiterungsmodul selbst ist in C geschrieben und bietet somit einen klaren Geschwindigkeitsvorteil gegenüber anderen SOAP-Stacks, die direkt in PHP implementiert sind.

2) *Probleme:* Dieses Vorhaben erwies sich allerdings als etwas problematisch. Das liegt einerseits an der knappen Dokumentation der SOAP-Erweiterung, welche ein Review des C-Quelltextes unerlässlich machte, und andererseits an einigen besonderen Features der SOAP-Erweiterung. Beispielsweise ruft der SOAP-Server anhand der Namen von Elementen im SOAP-Header automatisch Methoden gleichen Namens auf. Dies stellt eine potentielle Sicherheitslücke dar, weil damit auch Methoden aufgerufen werden können, für die keine WSDL-Ports spezifiziert wurden. Es muss daher eine Lösung gefunden werden, welche es ermöglicht die Header-Informationen aus der SOAP-Nachricht nach deren Verwendung zu entfernen. Die aktuelle Implementierung des SOAP-Servers hat noch einen weiteren Nachteil. Sie sendet nicht unter allen Umständen korrekte SOAP-Faults. Das senden si-

cherheitsrelevanter Fehler-Nachrichten sollte aber immer korrekt ablaufen, gerade wenn Exceptions zur Laufzeit auftreten.

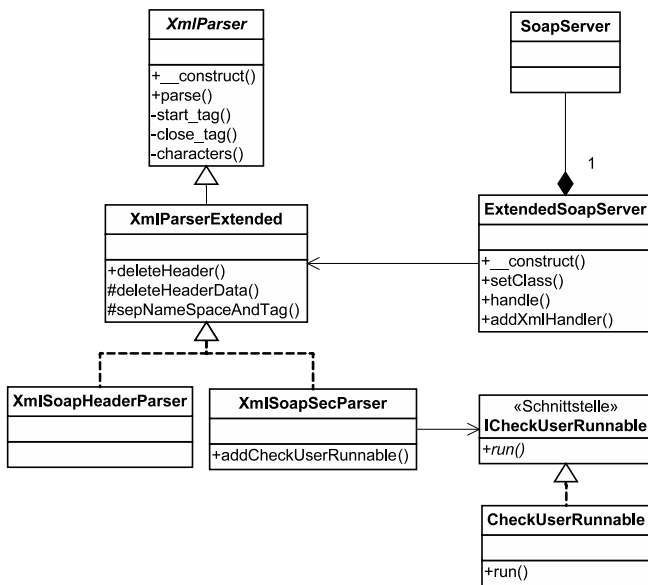


Abb. 6. Erweiterungen des SOAP-Servers für Web Services Security

3) *Extended SOAP-Server*: Um die Sicherheitsmechanismen in SOAP zu integrieren, wurde der „Extended SOAP-Server“ entwickelt.

Dabei war ein wichtiges Designziel, dass weitere Standards möglichst einfach integrierbar sind. Dazu wurde von dem Problem der Behandlung von Sicherheitsaspekten abstrahiert.

- *XmlParser*

Bei Web Services handelt es sich um eine Übertragung von XML-Nachrichten. Daher ist es offensichtlich, dass die Implementierung von XML-Handlern die Umsetzung von Web Service Standards massiv erleichtern kann. Es gibt zwei grundsätzlich verschiedene Arten ein XML-Dokument zu parsen. Auf dem Document Object Model (DOM) basierende Parser bilden im Speicher die Baumstruktur, des XML-Dokuments nach und ermöglichen somit den wahlfreien Zugriff auf alle Elemente. Die andere Möglichkeit sind sequentielle, Ereignis-getriebene SAX-Parser. Sie sind einfach beschrieben Tagsucher, welche bei jedem vorkommenden Tag eine Methode aufrufen. Für das Parsen von SOAP-Nachrichten empfehlen sich dabei klar die SAX-Parser, da sie performanter sind und speichereffizienter agieren. Eine abstrakte Implementierung eines solchen Parsers, welcher Methodensignaturen definiert und eine Methode `parse()` bereitstellt, ist der `XmlParser`. Diese Methode hat als Rückgabewert einen Fehlercode, der angibt, ob die Abarbeitung und Auswertung des Parsens erfolgreich war (`return 0`).

- *XmlParserExtended*

Der `XmlParser` wird durch `XmlParserExtended` erweitert. Da jeder Parser das gesamte Dokument durchsucht, ist es sinnvoll schon benutzte Elemente zu entfernen, um doppelte Datenauswertungen zu verhindern. Ebenso ist es möglich, durch die richtige Implementation der Methode `deleteHeader()` das Sicherheits-

problem der Ausführung von Methoden durch Header-Elemente zu verhindern (siehe V-D.6 *XmlSoapHeaderParser*).

Die XML-SOAP-Parser sind demnach eine Anwendung des Strategy-Patterns², denn sie erlauben es, mehrere Algorithmen unter einem einheitlichen Interface anzusprechen und auszutauschen. Damit können Handlerchains aus mehreren SOAP-Intermediaries realisiert werden, denn es ist möglich, dem Extended SOAP-Server beliebig viele XML-SOAP-Parser zu übergeben und diese sequentiell der Reihenfolge nach abzuarbeiten.

Das Username Token Profile ist folglich nur als eine konkrete Umsetzung des `XmlParserExtended` (siehe V-D.4 *XmlSoapSecParser*) anzusehen. Dabei wird in der `parse()`-Methode auf das Interface `ICheckUserRunnable` verwiesen und es einer Implementierung dieser Schnittstelle überlassen, mit den geparsen Werten für Username, Password, Nonce und Created die die Anfrage auszuwerten. Nachdem die Authentifikation erfolgt ist, werden alle entsprechenden Elemente mit der `delete()`-Methode gelöscht.

Durch die Trennung von Parser und SOAP-Server ist die Lösung unabhängig vom implementierten Standard. Eine Umstellung auf andere Token Profiles oder die Implementierung weiterer Web Service Standards ist somit leicht möglich.

Damit eine Behandlung der SOAP-Nachricht noch vor der Weiterleitung an den eigentlichen SOAP-Server (Ultimate Receiver) erfolgen kann, wird die SOAP-Nachricht aus der globalen Variablen `$HTTP_RAW_POST_DATA` ausgelesen und falls nötig durch die Parser angepasst. Das Ergebnis wird wieder in dieser Variablen gespeichert, da die gekapselte SOAP-Server-Implementierung auf diese Variable zugreift.

Es soll hierbei erwähnt werden, dass die Reihenfolge der Parser einen entscheidende Bedeutung hat. Ein Parser, welcher den gesamten SOAP-Header entfernt, sollte erst zum Schluss ausgeführt werden. Da allerdings der Extended SOAP-Server keine Kenntnis über die Semantik der genutzten XML-Parser hat, ist der Entwickler dafür verantwortlich, die XML Parser in der richtigen Reihenfolge beim Extended SOAP-Server zu registrieren. Diese Reihenfolge wird bei der Ausführung beachtet.

Nachdem die anspruchsvolleren Probleme gelöst werden konnten, blieb nur noch die Frage nach den SOAP-Faults (siehe V-D.7 *SoapFaults*) offen. Als gute Lösung erwies sich die Verwendung eines weiteren SOAP-Servers für das Versenden von Fehlermeldungen. Dabei wird ein minimaler Web Service bereitgestellt, der nicht einmal Dienste anbietet, denn er dient nur zum Initialisieren des Servers. Mit diesem SOAP-Server ist es nun möglich korrekte SOAP-Faults zu erstellen.

Damit sind alle eingangs benannten Sicherheitsprobleme erfolgreich gelöst.

Allerdings ist es nun vom Design her nicht möglich einen Extended SOAP-Server als Ableitung von `SoapServer` zu implementieren, da zwei Exemplare eines SOAP-Servers benötigt werden. Daher kapselt der Extended SOAP-Server nun

²„Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.“ [9]

sämtliche Funktionen des SOAP-Servers und erweitert ihn um die zusätzlichen Bestandteile (siehe Abb. 6).

Sollten später die Probleme in der SOAP-Implementierung behoben werden, ist eine Umstellung auf Vererbung aber nicht grundsätzlich unmöglich. Da eine Trennung der XML-Behandlung erfolgte, ist dies sogar mit relativ wenig Aufwand möglich.

Die Verwendung des Extended SOAP-Servers ist nun nahezu identisch zu der des normalen SOAP-Servers. Es wurde lediglich eine Methode nach außen hinzugefügt: `addXmlHandler()` registriert entsprechend der Erläuterung des Chain-Handlers neue XML-SOAP-Parser und führt sie aus.

4) *XmlSoapSecParser*: Nachdem die Umsetzung eines Servers näher erläutert wurde, besteht nun die spannende Frage, wie der XML-SOAP-Parser für das Username Token Profile aussieht.

Der Parser sammelt aus dem SOAP-Header alle notwendigen Daten zusammen. Dazu zählen die Elemente `Username`, `Password`, `Nonce` und `Created`. Sind die benötigten Daten ermittelt, so werden die Daten an eine Implementierung des Interfaces `ICheckUserRunnable` übergeben. Dieses wird intern in der Methode `parse()` aufgerufen, die damit auch Fehler nach außen reichen kann.

Die schon oft erwähnte Methode `deleteHeader()` dient nun dazu den gesamten oder nur Teile eines Headers zu entfernen. Grundsätzlich ist es hiermit auch möglich den Inhalt der gesamten SOAP-Nachricht zu löschen, was aber wenig Sinn macht. Bei der Implementierung muss die Variable `$this->deleteTag` einfach mit dem betreffenden Tag gesetzt werden. Nach der `parse()` Methode wird dann automatisch eine Methode des XML Parsers durch den `ExtendedSoapServer` aufgerufen, welche den gesamten Inhalt des Tags löscht. Sollen keine Elemente des SOAP-Headers entfernt werden, so braucht die Variable `$this->deleteTag` nicht gesetzt zu werden. In der konkreten Implementierung des `XmlSoapSecParsers` wird der gesamte Inhalt des `Security-Elements` gelöscht.

5) *ICheckUserRunnable* und *CheckUserRunnable*: Die Trennung ist notwendig, da verschiedenste Datenhaltungstechniken zur Speicherung der für die Authentifikation benötigten Benutzerdaten zum Einsatz kommen können. Nur so ist gewährleistet, eine flexible Implementierung zu erstellen. In der Implementierung des Interfaces `ICheckUserRunnable` ist die eigentliche Logik des Standards enthalten. Hier wurde als Beispiel eine Implementierung für die Benutzerverwaltung von `tele-TASK` erstellt.

Um den Ablauf zu verdeutlichen wurde ein Petrinetz erstellt, welches die Reihenfolge der Überprüfungen verdeutlicht. Es besteht aus zwei Teilen um die Komplexität etwas zu verringern. (siehe Abb. 7 und 8)

Die Implementierung des Username Token Profiles benötigt eine relationale Datenbank. Diese wird durch `ADOdb` und der Klasse `CheckUserDB` gekapselt. Die entsprechende Datenbanktabelle besitzt zwei Spalten mit den Attributen für `Nonce` (`Primary Key`) und dem `Created-Zeitstempel`. Mit diesen Informationen lassen sich doppelte und zu alte Anfragen filtern.

Etwas problematisch ist, dass das `type`-Attribut im `Password-Element` durch den `PHP5 SOAP-Client` nicht ge-

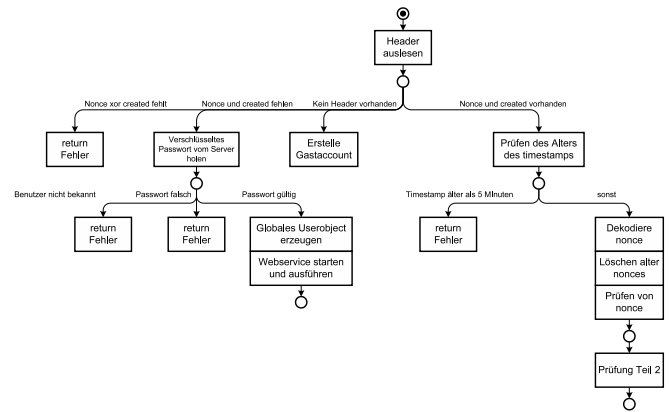


Abb. 7. Username Token Profile - Ablauf Teil 1

setzt werden kann. Um nun zu erkennen, ob `PasswordText` oder `PasswordDigest` verwendet wurde, wird in diesem Fall auf das Vorhandensein der Elemente `Nonce` und `Created` geachtet.

Nachdem der SOAP-Header ausgelesen und die notwendigen Daten übermittelt wurden, beginnt der Authentifikationsprozess, der in dem Petrinetz (Abb. 7 und Abb. 8) dargestellt ist. Es lassen sich mehrere Wege durch das Petrinetz finden. Von links nach rechts gilt:

- 1) Sollte entweder die Zufallszahl oder der Zeitstempel fehlen wird ein Fehler zurückgegeben. Dazu liefert die Methode `parse()` des `XmlSoapSecParser` nicht 0 sondern -101 (siehe V-D.7 *SoapFaults*)
- 2) Handelt es sich um eine Anfrage vom Typ `PasswordText`, dürfen weder `Nonce` noch `Created` im SOAP-Header auftauchen. Ist dies erfüllt findet die Authentifikation statt. Dabei kann es vorkommen, dass ein Benutzer unbekannt ist (-104) oder sein Passwort ungültig ist (-105). Bei einer erfolgreichen Authentifikation wird in unserer Beispielanwendung ein globales User-Objekt erzeugt, das später für die Autorisation verwendet werden kann. Anschließend wird die Anfrage an den Web Service weitergeleitet, sofern keine anderen Parser einen Fehler melden.
- 3) Der gesamte Security Header fehlt und es kann somit an dieser Stelle ein Gastaccount geladen werden.
- 4) Als letzter Zweig wird nun `PasswordDigest` abgearbeitet. Dafür wird als erstes das Alter der Anfrage überprüft. Dabei gilt, alle Anfragen älter als fünf Minuten zu verwerfen (-106). Sollte dies nicht der Fall sein wird die Zufallszahl dekodiert (Base64), alle alten Zufallszahlen werden entfernt und es wird überprüft ob diese Zahl schon vorhanden ist (-106).

Sollte nun die gestellte Anfrage nicht doppelt oder zu alt sein, so wird zuerst das Passwort (MD5 verschlüsselt) vom Server geholt. Sollte dies fehlschlagen ist hier ebenfalls der Benutzer unbekannt(-104). Mit dem nun erhaltenen Passwort wird auf Serverseite ein zweiter Vergleichshashwert berechnet und mit dem Wert aus der SOAP-Nachricht verglichen. Konnte der Hashwert korrekt erzeugt werden, so ist die Anfrage gültig und

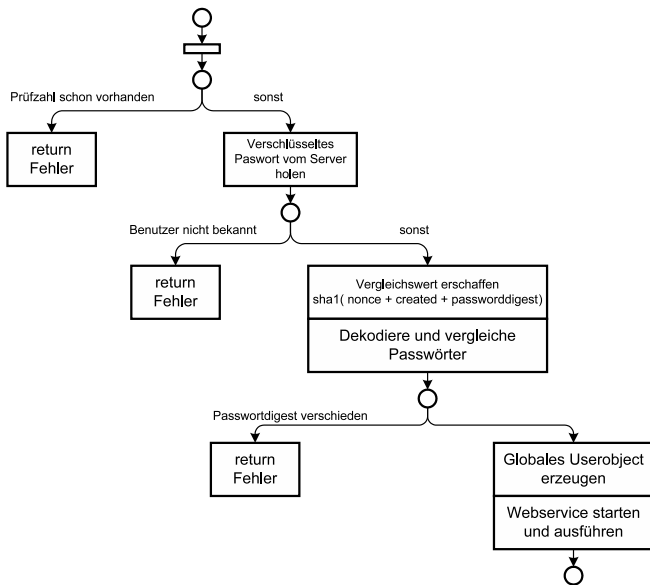


Abb. 8. Username Token Profile - Ablauf Teil 2

kann weitergereicht werden (Globales Userobject wird erzeugt und return 0 als Rückgabewert geliefert). Andernfalls wird auch hier ein Fehler zurückgegeben (-105)

6) *XmlSoapHeaderParser*: Dieser Parser löscht den gesamten SOAP-Header. Dazu nutzt er die Funktionalität des *XmlParserExtended* zum Entfernen von Header-Elementen. Dies funktioniert wie schon zuvor in *XmlSoapSecParser* beschrieben. Dabei ist der einzige Unterschied, dass nicht nur das Security-Tag entfernt wird, sondern sogar der gesamte SOAP-Header. Dieser Parser sollte daher als letzter an den Extended SOAP-Server übergeben werden.

7) *SOAP-Faults*: Die passenden SOAP-Faults für die jeweiligen Fehlersituationen sind ebenfalls im Web Services Security Standard spezifiziert (siehe [7] Soap Message Security 1.0). Tabelle I zeigt die internen Fehlercodes und die zugehörigen standardisierten SOAP-Faults sowie deren Beschreibung.

TABELLE I
FEHLERCODES

Interner Code	Fault-Code	Fault-String
-101	wsse:UnsupportedSecurityToken	An unsupported token was provided
-102	wsse:UnsupportedAlgorithm	An unsupported signature or encryption algorithm was used
-103	wsse:InvalidSecurity	An error was discovered processing the <wsse:Security> header
-104	wsse:InvalidSecurityToken	An invalid security token was provided
-105	wsse:FailedAuthentication	The security token could not be authenticated or authorized
-106	wsse:FailedCheck	The signature or decryption was invalid
-107	wsse:SecurityTokenUnavailable	Referenced security token could not be retrieved

8) *Fazit*: Als Abschluss soll noch einmal ein kritischer Blick auf die erstellte Lösung geworfen werden. Es sind alle geforderten Ziele umgesetzt worden. Die Benutzung des Extended SOAP-Servers unterscheidet sich nur durch die Anwendung von XML Parsern. Die WSDL-Beschreibung und die anwendungsspezifischen Klassen, welche die WSDL-Ports implementieren, bleiben dagegen völlig identisch.

Die unabhängige Implementierung der Sicherheitsaspekte konnte durch das Strategy-Pattern gelöst werden. Für eine Umsetzung von weiteren Web Services Standards wurde damit eine solide Grundlage geschaffen.

Etwas unschön ist lediglich, dass der Extended SOAP-Server nicht direkt von dem SOAP-Server der PHP-Erweiterung abgeleitet werden konnte. Falls zukünftige Versionen der SOAP-Implementierung dies wieder möglich machen, ist ein Refactoring des Extended SOAP-Servers mit sehr geringem Aufwand möglich.

Nachdem die Implementierung von Web Services Security und dem Username Token Profile auf Serverseite erklärt wurden, soll nun ein entsprechender SOAP-Client vorgestellt werden.

E. Umsetzung eines SOAP-Clients mit WS-Security

Das SOAP-Erweiterungsmodul von PHP5 stellt einen SOAP-Client zur Verfügung, der punktuell für das Username Token Profile erweitert wurde. Dabei konnte hier wie in Abb. 9 gezeigt von einer Vererbung profitiert werden. Aus der von vorhandenen Klasse *SoapClient* wurde der *SecureSoapClient* entwickelt.

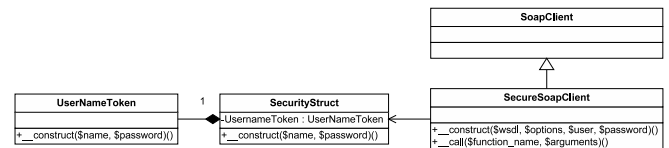


Abb. 9. Klassenstruktur des Clients

Dazu wurde der Konstruktor leicht verändert, damit die zusätzlichen Parameter wie Name und Passwort entgegengenommen werden können. Dieser bietet demnach nun 4 Parameter an in denen die zusätzlichen Informationen übergeben werden: `__construct($wsdl, $options, $user, $pw)`.

Mit den zusätzlichen Daten generiert der Client bei jedem Aufruf von `__call()` automatisch den benötigten SOAP-Header. Die Benutzung der Klasse ist dabei verglichen mit *SoapClient* nahezu identisch und unterscheidet sich lediglich im erweiterten Konstruktor. Das macht es Anwendern sehr leicht die Sicherheitsmechanismen zu integrieren.

Die Erstellung des SOAP-Headers ist allerdings mit der SOAP-Erweiterung von PHP5 etwas umständlich. Es ist erforderlich eine Struktur von Structs zu generieren, um einen konformen Header erzeugen zu können. Hierbei kommen auch die Klassen *SoapVar* und *SoapHeader* aus der SOAP-Erweiterung zum Einsatz. Der sogenannte *SecurityStruct*, der das Username Token kapselt, wird an *SoapVar* übergeben, welches daraus eine XML konforme Darstellung erzeugt. Das Ergebnis wird anschließend dem

SoapHeader übergeben. Dabei ist leider zu beachten, dass type-Attribute innerhalb eines Elements nicht erzeugt werden können. Damit ist die Interoperabilität vom PHP-Client zu beliebigen Web Services Security Servern leider nicht möglich. Aufgrund dessen, dass der Client nur PasswordDigest unterstützt, aber dies im Tag nicht identifizieren kann, wird er von anderen Servern als PasswordText interpretiert und dies sorgt für eine Ablehnung der Anfrage. Bei dem erstellten Extended SOAP-Server ist dies kein Problem, da er dies anhand der übergebenen Argumente herleitet und somit der Server einfach und problemlos mit dem Client kommunizieren kann. Demnach ist ein Erfolg bei anderen Servern abhängig von deren Implementierung.

Im Prinzip ist die Implementierung des Clients recht unspektakulär. Jedoch wurden einige undokumentierte Funktionen der SOAP-Erweiterung benutzt, was die Entwicklung weitaus schwieriger gestaltete, als anfänglich angenommen wurde.

Dieser Client kann durchaus als Vorlage für die Implementierung weiterer Token Profiles dienen. Eine Umstellung auf PasswordText wäre relativ problemlos möglich, um Interoperabilität zu anderen Servern gewährleisten zu können.

F. Einbettung in das Gesamtsystem

Nachdem die Implementierungen von Server und Client vorgestellt wurden, soll nun ein Blick auf das Gesamtsystem geworfen werden. Dabei wird das aus [5] bekannte Modell zu Grunde gelegt (siehe Abb. 10).

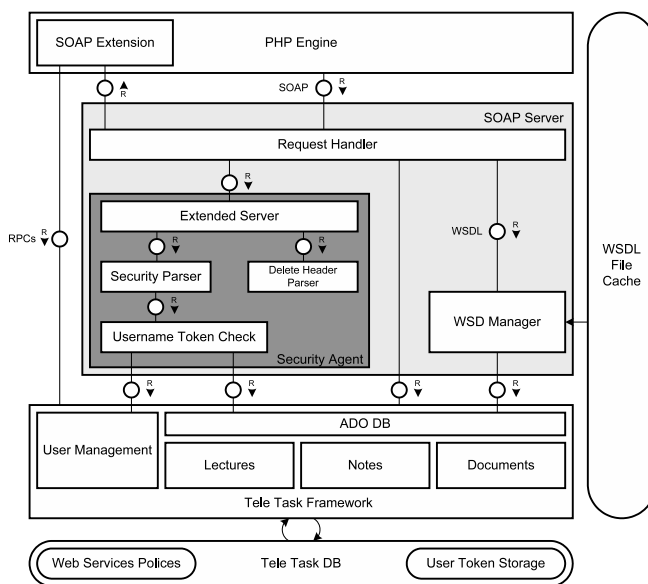


Abb. 10. Gesamtaufbau mit Sicherheit

In diesem Aufbaumodell wurde nun der Security Agent detaillierter dargestellt. Der Extended Server übernimmt dabei die Kapselung des eigentlichen SOAP-Server und leitet die Anfrage an die zwei Parser weiter, die damit entsprechend ihrer Funktion umgehen. Als letzter neuer Bestandteil übernimmt User Token Check die Authentifikation mit Hilfe der Datenbank und dem User Management.

Das Administrations-Tool, das in Kapitel VII näher erläutert wird, wurde so erweitert, dass es mit einem zusätzlichen Flag sichere Web Services erstellen kann.

G. Interoperabilität

Um die Interoperabilität zu überprüfen, wurde der sichere Web Service mit einem C# und einem Java Client getestet. Beide Interoperabilitätstests verliefen erfolgreich. Die Umsetzung des User Token Profiles auf Client Seite wird hier nicht näher erläutert. Es sei hier auf die Referenzimplementierungen³ verwiesen.

VI. RESTFUL WEB SERVICES

A. Representational State Transfer (REST)

Der Begriff REST bzw. Representational State Transfer geht auf Roy Thomas Fielding zurück und wird in [10] im Zusammenhang mit Web Services näher beleuchtet. Für Web Services im Allgemeinen ist eine Implementierung des REST-Architekturstils in Verbindung mit dem *Hypertext Transfer Protocol* (HTTP) besonders interessant und soll hier kurz aufgegriffen werden.

Der Hauptgedanke bei RESTful Web Services liegt in der Verwendung eines uniformen und minimalen Interfaces für alle Web Services, unabhängig von einem speziellen Einsatzbereich. Dies steht damit im Gegensatz zu den auf SOAP aufbauenden Web Services, welche als gemeinsame Basis nur das SOAP-Nachrichtenformat verwenden und darauf aufbauend jeweils eigene anwendungsspezifische Nachrichtenprotokolle und Schnittstellen definieren.

Wie in [10] dargelegt, hat es verschiedene Vorteile, eine allgemeingültige Schnittstelle zu verwenden, unter anderem wird damit die Interoperabilität verbessert, da unabhängig von einer speziellen Implementierung Aussagen über das zu erwartende Verhalten getroffen werden können. In Bezug auf HTTP-REST ergeben sich außerdem Vorteile bei der Verwendung von URI (Uniform Resource Identifier) bezogenen Fähigkeiten von diversen bestehenden Standards (z.B. XML-Dialekte), da so die Ressourcen eines RESTful Web Services ohne Umwege direkt in diesen Anwendungen verwendet werden können. Bei SOAP-basierten Diensten ist für solch eine Verwendung bisher immer ein Zwischenschritt nötig, da die Ressourcen dieser Dienste in einem eigenen Namensraum definiert sind.

B. Realisierung mit Hilfe des Toolkits

Für die Erstellung eines Web Services nach dem REST-Architekturstil sind verschiedene konzeptionelle Schritte nötig, die im Tutorial [11] und in [10] erläutert werden.

Prinzipiell sollte man sich an dieser Stelle des Konzepts einer Remote-Facade [12] bedienen, welche das System nach außen hin in geeigneter Weise kapselt und die Methoden zum Abfragen und Manipulieren der Ressourcen des Dienstes bereitstellt.

Das Toolkit stellt nun die benötigten Mechanismen bereit, um den entworfenen URI-Namensraum des REST Services auf

³Microsoft Web Services Enhancements 3.0 www.microsoft.com und Java Axis 1.3 <http://ws.apache.org/axis/>

diese Methoden abzubilden und die Ressourcenrepräsentationen in geeigneter Weise wählen zu können. Dafür lässt sich über einen regulären Ausdruck der Aufbau der URIs angeben, welche den Ressourcen entsprechen, die durch eine Methode repräsentiert werden. Für jede dieser Methoden kann außerdem der passende Serializer sowie Deserializer angegeben werden, um die Anfragedaten in programmiersprachliche Konstrukte bzw. in ein gewähltes Datenformat zu überführen.

Als Ergebnis wird mit Hilfe eines Tools aus diesen Angaben heraus ein *Deployment Descriptor* generiert, welcher von dem durch das Toolkit bereitgestellten `RESTServer` interpretiert wird, um die Anfragen an den Service abzuarbeiten.

C. Sicherheit für RESTful Web Services

Da diese Form der Web Services allein auf HTTP aufsetzt, kann hier ohne weitere Schwierigkeiten auf die Standardmechanismen zur Gewährleistung von Sicherheit zurückgegriffen werden. Sei dies nun mit *HTTP over TLS* (HTTPS) [13] die Verschlüsselung der Datenübertragung oder die Basic bzw. Digest Authentifizierungsmechanismen [14] zum Steuern des Zugriffs auf den Service. HTTPS ist in den meisten Fällen anwendungsunabhängig über den HTTP-Server realisierbar, für die Authentifizierung hingegen ist oft eine direkte Einbindung in die Anwendung wünschenswert.

Diese Anbindung erfolgt über die Implementierung des `AuthProvider`-Interfaces. Die kann dann über eine entsprechende Konfiguration des `RESTServer`s verwendet werden. Je nach gewünschtem Authentifizierungsverfahren gilt es hier darauf zu achten, dass gegebenenfalls die Nutzerpasswörter in einer geeigneten Art und Weise abgelegt werden. Details dazu sind in der Beschreibung des Interfaces bzw. im entsprechenden Abschnitt des Tutorials zu finden.

VII. ADMINISTRATIONS-TOOL & POLICY-PLUGIN

Das Administrations-Tool ist eine grafische Oberfläche, die der einfachen Konfiguration und Verwendung der Werkzeuge des Web Service Frameworks dient. Das Policy-PlugIn hingegen fungiert als Filter für Web Service Klassen bzw. deren Methoden.

A. Administrations-Tool

Das Administrations-Tool soll dem Benutzer die Handhabung des Web Service Frameworks erleichtern. Das Web Service Framework stellt eine Reihe nützlicher Programme zur Verfügung, die völlig unabhängig voneinander eingesetzt werden können. Hier setzt das Administrations-Tool an, dass eine intuitive und zentrale Möglichkeit der Administration und Konfiguration der Werkzeuge bieten soll.

Das Administrations-Tool nutzt eine eigene Bibliothek, die die grundlegende Funktionalität zur Verfügung stellt und die Anbindung an die externen Tools kapselt. Darüber hinaus wird für die Anbindung der Datenbank `ADOdb` genutzt und das Benutzerinterface wird mit Hilfe der `Smarty Template Engine` realisiert.

Allgemein hat der Nutzer die Möglichkeit Klassen beim Administrations-Tool zu registrieren und diese anschließend



Abb. 11. Grafische Oberfläche des Administrations-Tools

zu konfigurieren. Klassen werden entweder automatisch in einem gegebenen Klassenpfad gesucht oder können einzeln angegeben werden. Dabei kann schon der Programmierer einer Web Service Klasse mit Hilfe der Annotation `@webservice` seine Klasse als Web Service Klasse markieren. So markierte Klassen können vom Administrations-Tool herausgefiltert bzw. explizit gesucht werden. Zu einer registrierten Klasse kann nun angegeben werden, welche ihrer Methoden letztendlich veröffentlicht werden sollen. Auch dafür gibt es eine Annotation (`@webmethod`). Des Weiteren hat der Administrator die Möglichkeit sich Kommentare zu den Klassen und Methoden aus dem Quellcode anzeigen zu lassen. Zu diesen kann er zusätzlich eigene Kommentare hinzufügen, die dann von anderen Programmen weiterverwendet werden können. Eines dieser Programme ist der WSDL-Generator, der vom Administrations-Tool genutzt wird um WSDL-Dateien zu erzeugen (siehe Kapitel III). Diesem können die eigenen Kommentare zur Beschreibung von Web Service Klassen und Methoden übergeben werden. Ein weiteres Programm, das vom Administrations-Tool genutzt wird, ist der Adapter-Generator, der zur Erzeugung von speziellen Adapter-Klassen für Document/Wrapped Web Services dient (siehe Kapitel IV).

Letztendlich erzeugt das Administrations-Tool mit Hilfe der Generatoren und der Informationen zu den Web Service Klassen die jeweils notwendigen Dateien zur Bereitstellung eines Web Services und legt diese in dem vom Benutzer gewünschten Verzeichnis im Dateisystem ab.

Das Administrations-Tool bietet auch einen Assistenten (Wizard), mit dessen Hilfe Web Services im Frage-Antwort-Stil aufgesetzt werden können.

B. Policy-PlugIn

Wie schon erwähnt, dient das Policy-PlugIn der Filterung von Web Service Methoden. Hauptnutzer des PlugIns ist der WSDL-Generator. Dieser bekommt mit Hilfe des PlugIns diejenigen Methoden einer Klasse heraus, die zur Veröffentlichung mittels Web Service gedacht sind. Auch das Policy-PlugIn benötigt eine Datenbankbindung, welche wiederum durch `ADOdb` gekapselt wird. Durch das PlugIn werden in der Datenbank Informationen zu Klassen, deren Methoden und Kommentare hinterlegt. Schon durch die Benutzung des PlugIns wird eine Klasse samt ihrer Methoden registriert und kann dann wie beschrieben konfiguriert werden.

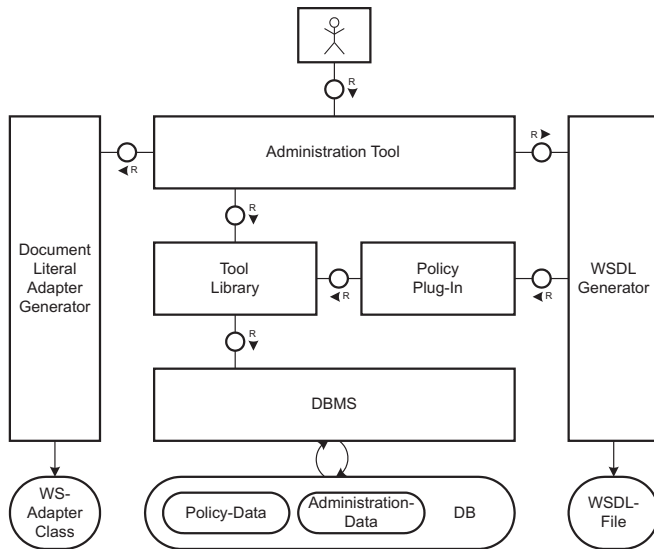


Abb. 12. Aufbaudiagramm des Administrations-Tools

VIII. ANWENDUNGSBEISPIEL

Parallel zur Erstellung des Web Service Toolkits wurde eine Beispielanwendung entwickelt, die auf der Datenbank des Teleteaching-Anbieters tele-TASK aufbaut. Diese Beispielimplementierung zeigt den Einsatz von Web Services und ermöglicht das Manipulieren der tele-TASK-Datenbank. Diesbezüglich stehen allerdings nur vorgegebene Anfragemöglichkeiten zur Verfügung. Die einzigen zu übergebenen Elemente sind Objekte des entsprechenden Typs, wie z.B. Lectures oder News. Aus diesen Klassen können WSDL-Dateien erzeugt und dann auf Funktionalität getestet werden. Dafür ist eine Kommentierung strikt nach dem Style Guide (siehe [5]) erforderlich. Sonst kann die Extended Reflection API die programmiersprachlichen Elemente nicht korrekt identifizieren und ohne diese Informationen kann der WSDL-Generator keine WSDL-Beschreibung für den Web Service erzeugen. Für eine korrekte Implementierung der auf ADOdb basierenden Datenbankabstraktionsschicht, welche mit einem Web Service angesprochen wird, müssen das Mapping und die Initialisierung durchgeführt werden.

A. Aufbau des Beispiels

Es werden zu jeder Datenbanktabelle zwei Klassen benötigt. Zum einen müssen die Datensätze (Entitäten) repräsentiert werden können. Dazu werden sie als Objekte behandelt, denn auf diese Weise lassen sich alle Werte mittels Get- und Set-Methoden adressieren. Zum anderen wird je eine Klasse benötigt, welche sich um den Datenbankzugriff kümmert und die gewünschten Anfragen auf der gesamten Tabelle durchführt. Im gegebenen Beispiel sind `Lecture` und `queryLecture` ein solches Klassenpärchen, wobei letztere die Datenbankfragen zur Verfügung stellt und abarbeitet. Dafür werden die Objekte aus der Anwendungslogik auf die entsprechenden Datensätze in der Datenbanktabelle gemappt.

B. Repräsentation der Datenbankentitäten

Die Klassen für die Datenbankentitäten, stellen die grundlegenden Elemente für die Datenbankzugriffsklasse bereit, indem die Objektattribute entweder mit Standardwerten, übergebenen Werten oder Daten aus der Datenbank initialisiert werden. In der Datenbank sind die Objekte derart hinterlegt, dass jedes Objekt einem Datensatz in der Datenbanktabelle entspricht. Zudem ist je eine Funktion, welche alle Werte auf den Standardwert zurücksetzen kann, eingefügt worden. Den größten Umfang nehmen die Get- und Set-Methoden ein, die für jedes Objektattribut, das lesbar und/oder modifizierbar ist, angelegt wurden.

C. Abfragen der Datenbank

Für lesende oder modifizierende Zugriffe auf die Datenbank wird der entsprechenden Methode der Datenbankzugriffsklasse der benötigte Datenblock übergeben und nach der Anfrage das Ergebnis in der erwarteten Form zurückgeliefert. Um dies zu ermöglichen, muss für jede denkbare, sinnvolle Anfrage eine Methode mit der gewünschten Funktionalität vorliegen. In der Klasse `queryLecture` ist es beispielsweise sinnvoll alle Vorlesungs-Elemente, alle Vorlesungen eines bestimmten Autors oder eines bestimmten Themas zurückzugeben. Zudem wird eine Methode benötigt, um Instanzen der Zielklasse zu erzeugen, damit der Zugriff auf die Datenbank serialisierbar wird. Im gegebenen Beispiel ist die Erzeugung des Rückgabewertes der aus Gründen der Übersichtlichkeit in einer weiteren Methode gekapselt worden. Dabei werden Rückgabewerte der Datenbank abgefangen und auf die Objekte abgebildet, so dass diese abschließend zur Verfügung gestellt werden können.

IX. ZUSAMMENFASSUNG UND AUSBLICK

Die von uns entwickelte Erweiterung der PHP5 Reflection API ermöglichte uns auf äußerst elegante Weise einen WSDL-Generator zu implementieren. Der entscheidende Vorteil dabei ist, dass die Algorithmen zum Ermitteln der Typinformationen sich nicht direkt im WSDL-Generator befinden. Dadurch konnten wir gleich eine ganze Familie von weiteren Web Service Werkzeugen auf Basis der Extended Reflection API entwickeln. Die Fähigkeiten der Extended Reflection API sind aber auch für viele weitere PHP Projekte interessant. Der eingeführte Annotation-Mechanismus könnte zudem eine Grundlage für die Nutzung von neuen dynamischen Programmier-Techniken wie zum Beispiel aspekt-orientierter Programmierung in PHP sein.

Der WSDL-Generator unterstützt den WSDL 1.1 Standard für Document/Literal, RPC/Literal und RPC/Encoded Web Services. Zur Markierung von Klassen und Methoden als Web Service können Annotationen ähnlich wie in Java EE und .NET verwendet werden. Zusätzlich werden semantische Informationen aus dem Quelltext in die WSDL-Dokumentation übernommen. Für Document/Wrapped Web Services können mit dem Adapter-Generator automatisch Adapter-Klassen generiert werden, die sich transparent um das Wrapping in den SOAP-Nachrichten kümmern.

Die rund 20 weiterführenden WS-*-Spezifikationen im Bereich der Sicherheit, Synchronisation und Sessioning, an denen zur Zeit gearbeitet wird, werden zusätzliche Felder für die Header von SOAP-Nachrichten spezifizieren. Um diese Standards in Zukunft mit PHP nutzen zu können entwickelten wir den Extended SOAP Server, der um weitere Handler erweitert werden kann, die als Kette von SOAP-Intermediaries nacheinander eine SOAP-Nachricht bearbeiten. Auf der Basis des Extended SOAP Servers haben wir das Username-Token-Profile aus dem WS-Security Standard für PHP5 implementiert. Damit ist es Anwendungen, die einen Web Service nutzen, möglich sich beim Dienstanbieter sicher zu authentifizieren.

Als Alternative zu Web Services mit komplexen SOAP-Stacks und dem damit verbundenen Aufwand, werden die Web Services im REST-Stil besonders bei Entwicklern zunehmend beliebter. Dies hat auf der einen Seite mit dem geringen Toolbedarf und damit einem geringeren Einstiegsaufwand zu tun und auf der anderen Seite mit den Vorteilen dieses Architekturstils und der damit verbundenen Interoperabilität. Um dieser Entwicklung Rechnung zu tragen, wurden zusätzliche zu den SOAP-Tools entsprechende Werkzeuge zum Erleichtern der Entwicklung von REST Services implementiert. Diese ermöglichen die automatische Generierung eines REST Servers der die Serviceanfragen entgegen nimmt und außerdem bei Bedarf die Authentifikation des Clients über HTTP Digest durchführt.

Das entwickelte Administrations-Tool bietet schließlich eine einheitliche Oberfläche um alle Werkzeuge zu konfigurieren und zu starten und damit automatisiert SOAP-Server und REST-Server für eine bestehende Anwendung zu generieren.

Trotzdem es Pläne gibt die entwickelten Werkzeuge in ei-

nem konkreten Projekt einzusetzen, lag ein besonderes Augenmerk darauf alle Komponenten so allgemein und wiederverwendbar wie möglich zu gestalten.

REFERENZEN

- [1] *SOAP Version 1.2 specification*, W3C Recommendation World Wide Web Consortium (W3C), June 24, 2003. <http://www.w3.org/TR/soap/>
- [2] *Web Services Description Language (WSDL) 1.1*, W3C Note World Wide Web Consortium (W3C), March 15, 2001. <http://www.w3.org/TR/wsdl/>
- [3] *Universal Description, Discovery and Integration v3.0.2 (UDDI)* OASIS, Oktober 19, 2004. <http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm>
- [4] C. Hartmann, M. Sprengel, M. Perscheid, G. Gabrysiak, F. Menge, *Einführung in XML Web Services*, Ausarbeitung zum Seminar Konzepte und Methoden der Web-Programmierung WS2005/06, Hasso-Plattner-Institut für Softwaresystemtechnik, November 2005
- [5] S. Böttner, A. Meyer, S. Marr, *Web Service Facade for PHP5*, Ausarbeitung im Seminar Konzepte und Methoden der Web-Programmierung WS2005/06, Hasso-Plattner-Institut für Softwaresystemtechnik, 2006
- [6] Frank Cohen, *Discover SOAP encoding's impact on Web service performance*, März 2003. <http://www-128.ibm.com/developerworks/webservices/library/ws-soapenc/>
- [7] OASIS: www.oasis-open.org, 2006
- [8] *Username Token Profile*: <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf>, 2004
- [9] E. Gamma, R. Helm, R. Johnson und J. Vlissides *Design Patterns* Addison-Wesley, 1995
- [10] S. Marr, *RESTful Web Services*, Ausarbeitung zum ADT-Seminar WS2005/06, Hasso-Plattner-Institut für Softwaresystemtechnik, 2006
- [11] TUTORIAL LINK
- [12] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford, *Patterns of Enterprise Application Architecture*. Addison Wesley, 2002.
- [13] E. Rescorla. RFC 2818: *HTTP Over TLS*. Internet Engineering Task Force, Mai 2000.
- [14] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart. RFC 2617: *HTTP Authentication: Basic and Digest Access Authentication*. Internet Engineering Task Force, Juni 1999.