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 Administrations-Tool stellt eine einheitliche grafische Oberfläche für alle Werkzeuge zur Verfügung um automatisiert SOAP-Server und REST-Server für eine bestehende Anwendung zu generieren und somit plattformübergreifende Kommunikation mit anderen Anwendungen zu ermöglichen.**

***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 Produkte 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 Java EE gewohnt ist. Zwar ist die SOAP-Implementierung sehr einfach zu benutzen, aber die manuelle Erstellung von WSDL-Beschreibungen ist 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, mit denen die Web Services sofort bereit für den Produktiveinsatz sind.

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. Dannach 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 die in einer Programmiersprache formulierten Informationen zu kommen, welche die

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

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.

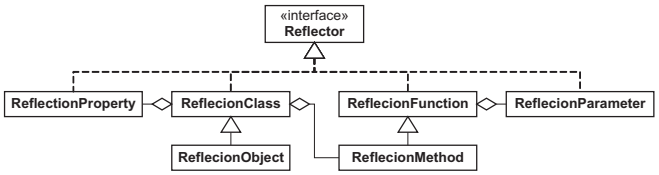

Abb. 1. UML-Darstellung der PHP5 Reflection API

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 diesen 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 zur dessen Laufzeit, sondern auch ohne ein Exemplar des Konstrukts, die gewünschten Typinformationen zu gewinnen.

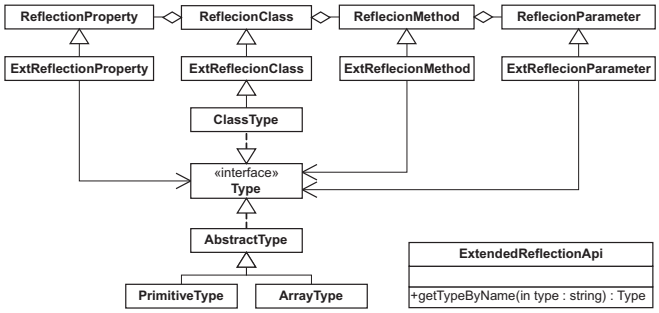

Abb. 2. UML-Darstellung der erweiterten Reflection API

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 Methodenaufrufsstack 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. Demzufolge muss nur die Nachricht benannt werden um die Parameterliste direkt übergeben zu können, da die Nachricht deren Art und Reihenfolge bereits vorgibt. 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. Die erste Variante ist es eine spezielle Kodierung zu verwenden. Die einzige gebräuchliche Kodierung ist das im SOAP-Standard [1] spezifizierte SOAP-Encoding. Diese schreibt vor, wie man alle Typen der jeweiligen Programmiersprache, auf vom Standard definierte 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.

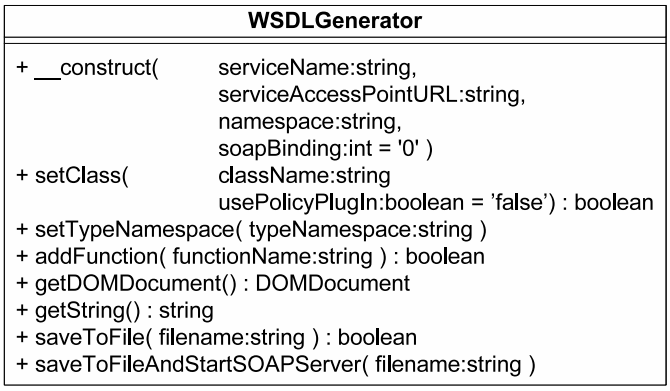

Abb. 3. Klassendiagramm des WSDL-Generators

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 uns nämlich nicht nur die Methodennamen, sondern wir erhalten Listen von ExtReflectionMethods, die wir direkt nach ihren Parametern und Rückgabewerten fragen können. Dies ist einer der Hauptvorteile des `WSDLGenerators`, da er nicht selbst die Klassen parsen muss, um die benötigten Informationen zu erhalten.

Jede 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 in Abschnitt VII-B vorgestellt wird, genutzt werden. Es erhält als Eingabe die Liste der ExtReflectionMethoden und liefert eine gefilterte Liste zurück. Die Entscheidung, 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 erzeugt 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 erzeugten Schemanamen in einer Liste und greift während der Verarbeitung auch auf diese zu.

Der Generator erzeugt ein DOM-Dokument, das die WSDL-Struktur enthält. Das heißt im Einzelnen: Es gibt das Definition-Element, in dem alle verwendeten Namespaces definiert werden und das alle anderen Beschreibungselemente enthält. Dann folgt die Type-Sektion, die XML Schema Beschreibungen für alle komplexen Datentypen enthält und, bei Verwendung einer Literalkombination, 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 angegeben in der Binding-Sektion und abschließend kommt noch 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 gerufenen Methode oder Funktion ausgeführt werden muss. Auf der einen Seite bietet dieses Vorgehen 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 normale 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 normalen 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 statt der Original-Klasse beim SOAP-Server registriert. Die Methoden der Adapter-Klasse rufen mit den Argumenten aus dem jeweiligen vom SOAP-Server übergebenen Objekt die Methode an der Orginal-Klasse auf und verpacken den Rückgabewert in ein Array bevor sie ihn zur Serialisierung an den SOAP-Server zurückgeben.

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 und die Orginal-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.

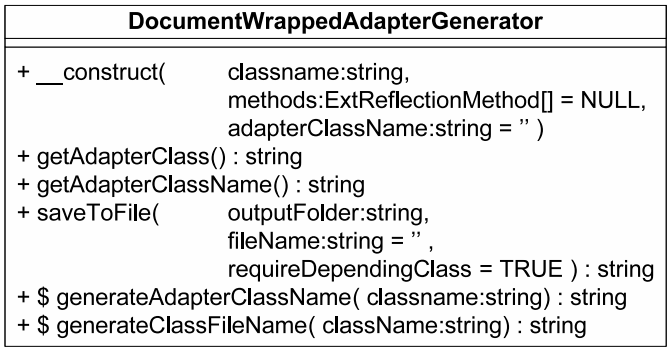

Abb. 4. Klassendiagramm des Document/Wrapped-Adapter-Generators

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

V. WS-Security

A. Übersicht

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 Spezifikationen von Web Services in ihrem normalen Umfang von WSDL, SOAP 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. Die Frage nach Sicherheitsmechanismen für Web Services steht nun im Zeichen dieses Kapitels.

B. Anforderungen an gesicherte Webservices

Wird allgemein von Sicherheit gesprochen, sind viele Dinge zu beachten. Bei Sicherheit handelt es sich meist nicht nur um funktionale, sondern auch um nicht-funktionale Anforderungen. Um Sicherheit in Software untersuchen zu können, existieren verschiedenen Kriterien. Im Allgemeinen sind dabei die Punkte Verfügbarkeit, Datenintegrität, Vertraulichkeit, Authentifikation und Autorisation anerkannt. Diese Aspekte werden hier im Weiteren nicht näher erläutert, sollen aber auf eine gewisse Sensibilität aufmerksam machen, die bei einer Umsetzung von Sicherheit gewährleistet werden muss.

Für eine Umsetzung von Sicherheitsaspekten bei der Verwendung von Web Services müssen vor Beginn der Arbeiten einige Designziele aufgestellt werden, die es einzuhalten gilt. Diese beeinflussen die Lösungsfindung entscheidend. Hierzu zählt vor allem, dass die Implementierung bzw. die Benutzung von Sicherheitsaspekten nicht zu Lasten des Anwendungsentwicklers gehen darf. Die Implementierung eines Web Services muss vollständig unabhängig von möglichen Sicherheitstechniken sein. Dies liegt darin begründet, dass eine sichere Implementierung morgen schon als sehr unsicher gelten kann. Aus diesem Grund darf eine Sicherheits-Implementierung nur eine Erweiterung zur Business-Logik sein, welche schnell ausgetauscht werden kann. Ein weiterer Punkt betrifft eine möglichst große Interoperabilität. Auch wenn PHP nicht die klassische Software-Plattform für die Implementierung von Web Services ist, so muss eine Lösung dennoch aus Java und .Net genutzt werden können.

Das Thema Sicherheit umfasst einen riesigen Themenkomplex. Daher ist es nicht möglich, eine vollständige Sicherheitslösung in einem kurzen Zeitraum umzusetzen. Hierbei sind auch viele Reviews nötig, um wirkliche Sicherheit zu gewährleisten. Es musste daher auf einen Teil der zuvor genannten Sicherheitsaspekte das Hauptaugenmerk gelegt werden. Dies soll im Nachfolgenden bei der Umsetzung die Authentifizierung sein. Die Web Services werden dabei weiterhin unverschlüsselt übertragen. Allerdings dürfen andere Aspekte nicht außer Acht gelassen werden, so sollte z.B. die entstehende Lösung weiteren Sicherheitsstrategien nicht im Wege stehen.

C. Mögliche Ansätze zur Absicherung von Web Services

Möchte man hier eine Lösung finden, fällt gleich ein grober Ablauf ein. Dieser besteht aus:

1) Anmelden, sowie Verschlüsseln der Anmeldedaten auf Clientseite
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 Service Security[1] von Oasis eine weite Verbreitung gefunden. Dieser Standard umfasst mehrere Teile. Das ist auf der einen Seite Soap Message Security zum Verschlüsseln von Nachrichten und der Behandlung von Fehlern. Auf der anderen Seite gibt es die Token Profiles[2]. Diese beschreiben einen Authentifizierungsmechanismus für Web Services. Sie erfüllen dabei idealerweise fast alle geforderten Ziele und wurden daher für

[1]Basiert auf Standard 1.0, am 17.2.2006 wurde die Version 1.1 veröffentlicht

[2]Hier wird nur das Username Token Profil 1.0 betrachtet

eine Implementierung in PHP ausgewählt. Eine Umsetzung des Username Token Profile 1.0 steht nun im Mittelpunkt der weiteren Arbeiten.

D. Username Token Profile 1.0 im Detail

Das Username Token Profile 1.0 wurde von Oasis [7] erarbeitet. In der hier vorgestellten Version, handelt es sich, wie schon erwähnt nicht um die Version 1.1, da sie zu Beginn der Entwicklung nur als Entwurf vorlag und der erweiterte Funktionsumfang nicht benötigt wurde. Des Weiteren wird Version 1.0 schon durch diverse Middleware-Plattformen unterstützt und bietet somit eine gute Grundlage für Interoperabilitätstest und Anwendungen. Der Standard beschreibt nicht neue Techniken, sondern gibt einen Rahmen vor, wie die benötigten Informationen in einem spezifizierten Format zu übertragen sind. Dabei wird kein Verschlüsselungsalgorithmus für das Passwort festgelegt, da dessen Wert kontinuirlich mit der Zeit abnimmt.

Das Format besteht aus dem ersten Tag `<wsse:Security>`, welches die Webservice Security Daten im SoapHeader kapselt. Darauf folgt ein Tag namens `<wsse:UsernameToken>`, der darauf aufmerksam macht, dass hier das Username Token Profil benutzt wird. Nun folgen die benötigten Daten, dabei gilt, `<wsse:Username>` muss immer angegeben werden und ist unverschlüsselt. Beim nächsten Tag `<wsse:Password Type="...">` unterscheidet man zwei Typen:

- *PasswortText (default)*
  Hierbei kann das type Feld leer gelassen werden und als Daten sind nur der Username und das Passwort (bzw. Passwortäquivalent) notwendig.
- *PasswordDigest*
  Um sicherzustellen, dass keine Anfrage zweimal gesendet werden kann (Replayattacke), wird das Passwort mittels eines Zeitstempels und einer Zufallszahl (Nonce) verschlüsselt. Als `Type`-Wert muss hier grundsätzlich PasswordDigest gesetzt sein. Die Formel nach welcher sich die Verschlüsselung zusammensetzt, sieht nun wie folgt aus:

  ```
  PasswordDigest = Base64( SHA-1( Nonce
  + Created + Password ))
  ```

  Die Zufallszahl wird mit dem Datum und dem Passwort (bzw. Äquivalent) zusammengekettet, danach wird mittels eines SHA-1-Algorithmus dieser Wert gehasht und nochmals mit Base64 kodiert. Auf Serverseite wird mittels der bekannten Daten (Zufallszahl und Zeitstempel sind im Header enthalten) ebenfalls ein Vergleichswert erschaffen, um damit dann die Korrektheit der Anfrage zu überprüfen. Somit ist jede Anfrage nur genau einmal gültig (Hashwert ist jedes Mal verschieden). Identische Anfragen werden natürlich von der Serverseite negiert.

Die letzten zwei Tags sind optional und senden zum einen die Zufallszahl(`<wsse:Nonce>`) im Base64 Format und zum anderen den Erzeugungszeitpunkt(`<wsu:Created>`) mit.

```
Nonce = Base64( Nonce )
```

Bei den Tags werden nun zwei Namespaces sichtbar, dabei handelt es sich einmal um wsse[3] und zum anderem um wsu[4].

Des Weiteren sollen Anfragen mit einem alten Zeitstempel zurückgewiesen und ein Cache mit schon benutzen Zufallszahlen geführt werden. 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 dieser Umsetzung gilt grundsätzlich das Passwörter immer MD5 gehasht sein müssen. Dementsprechend gilt auch beim PasswortText, dass ein Passwort einem `MD5(Passwort)` entspricht und nicht nur durch ein reines Passwort repräsentiert wird.

```
<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. Soap Header Beispiel Username Token Profil

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

E. Umsetzung des Servers in PHP

1) Vorhandene Umsetzungen: PHP bietet seit der Version 5 eine SOAP Extension, welche es erlaubt Web Services aufzusetzen, beziehungsweise angebotene Dienste zu verwenden. Eine Möglichkeit der Verwendung von Sicherheitsaspekten bleibt einem dabei allerdings verwehrt. Die Implementierung des Username Token Profiles für PHP wird auf Basis der vorhandenen Klassen SoapClient und SoapServer erfolgen. Dies hat den klaren Vorteil, dass der gesamte Teil zum Abarbeiten von Web Services schon als gelöst angesehen werden kann. Im Idealfall profitiert die zu erstellende Lösung einfach aus Verbesserungen der SOAP Extension. Die Extension selbst ist in C geschriebenen und bietet somit einen klaren Geschwindigkeitsvorteil gegenüber PHP Klassenbibliotheken. Die erstellten PHP Klassen erben von diesen bzw. erweitern sie um die zusätzlichen benötigten Aspekte.

2) Probleme: Diese Idealvorstellung erweist sich allerdings schnell als nicht trivial. Dies liegt einerseits in der schlechten PHP Dokumentation, welche ein Review des SourceCodes unerlässlich macht und andererseits in den noch vorhanden

[3]http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext- 1.0.xsd

[4]http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility- 1.0.xsd

„Features“. Unter anderem ruft der SoapServer anhand des Tagnamens im SoapHeader automatisch Methoden auf, welche den gleichen Namen besitzen. Dies kann Potential für eine Sicherheitslücke darstellen. Es muss daher eine Lösung gefunden werden, welche es ermöglicht alle Headerinformationen aus der SoapMessage nach deren Verwendung zu löschen. Die vorhandene Implementierung des SoapServers hat noch einen weiteren Nachteil. Sie wirft nicht unter allen Umständen korrekte SoapFaults. Das Werfen eigener Fehler sollte hingegen immer korrekt ablaufen, gerade wenn Exceptions zur Laufzeit auftreten.

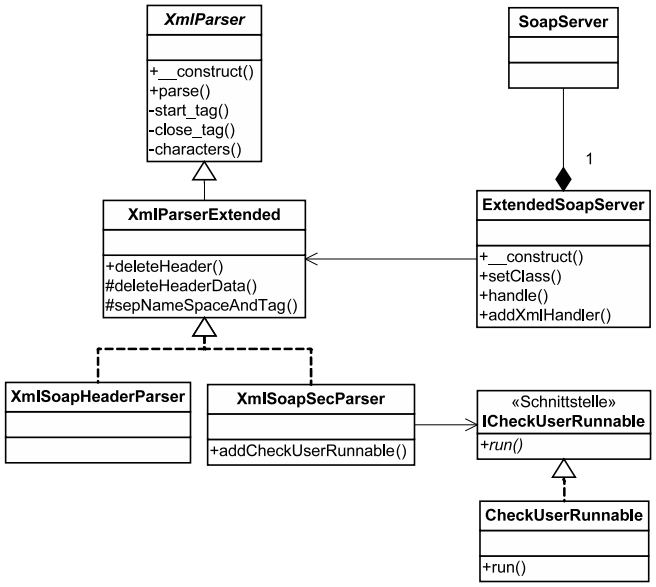

Abb. 6. Klassenstruktur der Webservice Security

3) Extended Soap Server: Um die Sicherheit kapseln zu können, wurde der „Extended Soap Server“ entwickelt.

Es sei an ein wichtiges Designziel erinnert: Weitere Standards sollten möglichst einfach integrierbar sein. Aus diesem Grund muss dass Problem der Behandlung von Sicherheitsaspekten abstrahiert werden. Dies ist hier geschehen und wird im Weiteren näher erläutert.

- *XmlParser*
  Bei Web Services handelt es sich um eine Übertragung von XML konformen Dokumenten. Da ist es nur offensichtlich, dass die Implementierung von XML Handlern den Weg zum Designziel massiv erleichtern können. Es gibt zwei grundsätzlich verschiedene Arten ein Dokument zu parsen. Einerseits die DOM-Parser, welche im Speicher die gesamte Baumstruktur aufbauen und andererseits die SAX-Parser (Event-Driven). Sie sind einfach beschrieben Tagsucher, welche bei jedem vorkommenden Tag eine Methode aufrufen. Für eine Umsetzung in PHP empfehlen sich dabei klar die SAX-Parser, da sie speichereffizienter agieren. Diese lassen sich in PHP nicht direkt realisieren, doch stehen Grundelemente bereit. Eine abstrakte Implementierung eines solchen Parsers, welcher leere Methoden implementiert, aber schon eine Methode `parse()` bereitstellt, ist der `XmlParser`. Diese Methode hat als Rückgabewert einen Fehlercode, welcher angibt ob die Abarbeitung und Auswertung des Parsens erfolgreich war (`return 0`).
- *XmlParserExtended*
  Der `XmlParser` wird durch `XmlParserExtended` erweitert, wodurch es möglich ist, Headerabschnitte zu löschen. Da jeder Parser das gesamte Dokument durchsucht, ist es nötig schon benutzte Teile zu entfernen, um doppelte Datenauswertungen zu verhindern. Somit ist es ebenso möglich, durch die richtige Implementation der Methode `deleteHeader()` das Sicherheitsproblem der Ausführung von Methoden durch Header-Tags zu verhindern (siehe V-E.6 *XmlSoapHeaderParser*).
  Die XML Soap Parser sind demnach eine Anwendung des Strategy-Pattern[5], welches es erlaubt mehrere Algorithmen unter einem einheitlichen Interface anzusprechen und auszutauschen. Mit dieser Möglichkeit kann ein Chainhandler aufgebaut werden. Dabei handelt es sich um die Möglichkeit, beliebig viele XML Soap Parser dem `ExtendedSoapServer` zu übergeben und diese nach der Reihenfolge abzuarbeiten.
  Das Username Token Profile ist folglich nur als eine konkrete Umsetzung des `XmlParserExtended` (siehe V-E.4 *XmlSoapSecParser*) anzusehen. Dabei wird in der `parse()`-Methode auf das Interface `ICheckUserRunnable` verwiesen und überlässt ihm mittels der geparsten Werte für Username, Passwort, Nonce und Created die Auswertung der Anfrage. Nachdem diese erfolgt ist, werden alle entsprechenden Tags mittels der `delete()`-Methode gelöscht.

Durch die Trennung von Parser und SoapServer ist die Lösung unabhängig vom Parser und somit dem zu implementierenden Standard. Eine Umstellung auf andere Möglichkeiten ist somit leicht möglich.

Damit eine Behandlung der SoapMessage noch vor der Weiterleitung an den PHP eigenen SoapServer erfolgen kann, muss der Inhalt aus `$HTTP_RAW_POST_DATA` ausgelesen und durch die Parser jeweils, wenn nötig, angepasst werden. Das Ergebnis muss wiederum in dieser Variable gespeichert werden, da die gekapselte SoapServer-Implementierung auf diese Variable zugreift.

Es soll hierbei erwähnt werden, dass die Reihenfolge der Parser einen entscheidenden Faktor spielen kann. Ein Parser, welcher den gesamten SoapHeader löscht, darf erst zum Schluß ausgeführt werden. Da allerdings der `ExtendedSoapServer` keine Kenntnis über den semantischen Inhalt der genutzten XML Parser hat, ist der Entwickler des `ExtendedSoapServer` dazu aufgefordert, die XML Parser in der richtigen Reihenfolge dem `ExtendedSoapServer` mitzuteilen. Diese Reihenfolge wird bei der Ausführung beachtet.

Nachdem die großen Probleme gelöst werden konnten, bleibt nur noch die Frage nach den SoapFaults(siehe V-E.7 *SoapFaults*). Als gute Lösung erwies sich in Tests die Verwendung eines extra SoapServers nur für Fehlermeldungen.

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

Dabei muss ein minimaler Web Service bereitgestellt werden, welcher nicht einmal Dienste anbieten muss. Dieser dient nur zum Aufsetzen des Servers. Damit ist es möglich korrekte SoapFaults zu erstellen. Die Verhinderung der genannten Sicherheitsprobleme ist damit erfolgt.

Allerdings ist es nun leider nicht mehr möglich einen `ExtendedSoapServer` als Ableitung von SoapServer zu implementieren, da nun zwei Exemplare eines SoapServers benötigt werden. Demnach kapselt der `ExtendedSoapServer` nun sämtliche Funktionen des SoapServers und erweitert ihn um unsere Bestandteile (siehe Abb. 6).

Sollten später die Probleme in PHP 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 `ExtendedSoapServers` ist nun nahezu identisch zu der des normalen SoapServers. Es wurde lediglich eine Funktion nach außen hinzugefügt: `addXmlHandler()` fügt entsprechend der Erläuterung des ChainHandlers neue XML Soap Parser hinzu 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 Header alle notwendigen Daten zusammen. Dazu zählen Username, Password, Nonce und das Created-Tag. Sind die benötigten Daten ermittelt, so werden die Daten an eine Implementierung des Interfaces `ICheckUserRunnable` übergeben. Dieses wird intern in der Methode `parse()` aufgerufen, welche damit auch die Fehler nach außen reichen kann.

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

5) ICheckUserRunnable und CheckUserRunnable: Die Trennung ist notwendig, da bei einer Anwendung in einem Unternehmen immer wieder verschiedene Datenhaltungstechniken verwendet werden. 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 tele-TASK-Benutzerverwaltung 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)

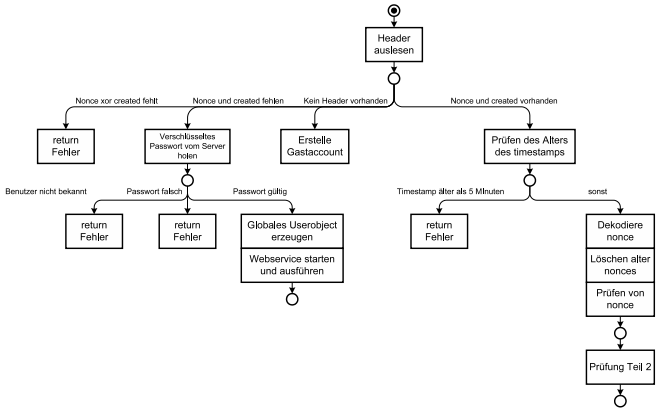

Abb. 7. Username Token Profile - Ablauf Teil 1

Vorweg sei gesagt, die Umsetzung des Username Token Profile benötigt eine eigene Tabelle. Diese wird durch ADOdb und der Klasse `CheckUserDB` gekapselt. Die Tabelle besitzt zwei Spalten mit den Attributen für Nonce (Primary Key) und dem Created Zeitstempel. Mittels derer lassen sich doppelte und zu alte Anfragen filtern.

Ein weiterer wichtiger Punkt ist, dass das `type`-Feld im Password Tag durch den PHP Soap Client nicht gesetzt werden kann. Um nun zu erkennen, ob `PasswordText` oder `PasswordDigest` verwendet wurde, wird in diesem Fall auf die gesetzten Tags geachtet.

Ablauf

Als erstes wird, wie schon oft beschrieben, der SoapHeader ausgelesen und die notwendigen Daten übermittelt. Es lassen sich mehrere Wege durch das Petrinetz finden, von links nach rechts gilt:

1) Sollte die Zufallszahl exklusiv oder der Zeitstempel fehlen wird ein Fehler zurückgegeben. Daher liefert die `parse()` Funktion des `XmlSoapSecParser` nicht 0 sondern -101 (siehe V-E.7 *SoapFaults*)
2) Hierbei handelt es sich um PasswordText, dafür müssen sowohl Nonce als auch Created fehlen. Anschließend findet die Authentifikation statt. Hierbei kann es passieren, dass ein Benutzer unbekannt ist (-104), sein Passwort ungültig ist (-105) oder im Erfolgsfall das globale Userobjekt erzeugt und die Anfrage an den Webservice weitergeleitet wird, sofern keine anderen Parser Fehler auslösen.
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

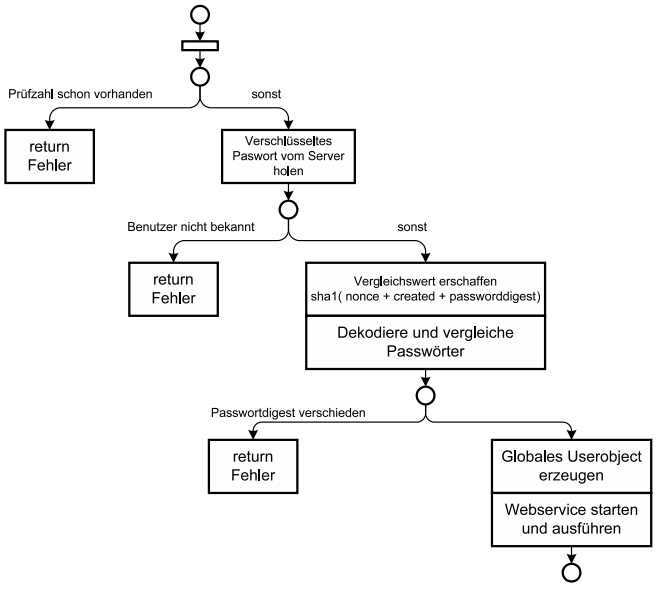

Abb. 8. Username Token Profile - Ablauf Teil 2

Vergleichshashwert erzeugt und mit dem Zugesandten verglichen. Konnte der Hashwert korrekt erzeugt werden, so ist die Anfrage gültig und kann weitergereicht werden (Globales Userobjekt wird erzeugt und `return 0` als Rückgabewert geliefert). Andernfalls wird auch hier ein Fehler zurückgegeben (-105)

6) XmlSoapHeaderParser: Dieser Parser löscht den gesamten SoapHeader. Demnach nutzt er die Funktionalität des `XmlParserExtended` zum Löschen von Tag-Abschitten. Dies funktioniert wie schon zuvor in `XmlSoapSecParser` beschrieben. Dabei ist der einzige Unterschied, dass nicht nur das Security-Tag entfernt wird, sondern gleich der gesamte SoapHeader. Dieser Parser sollte demnach als allerletztes an den `ExtendedSoapServer` übergeben werden.

7) SoapFaults: Das Thema SoapFaults basiert ebenfalls auf dem Web Service Security Standard (siehe [7] Soap Message Security 1.0) und ist somit wohl definiert. Die folgende Tabelle I zeigt den internen Fehlercode und den zugehörigen standardisierten Fehlercode sowie dessen Stringbeschreibung.

8) Fazit: Als Abschluss soll noch einmal ein kritischer Blick auf die erstellte Lösung geworfen werden. Dabei sind viele der geforderten Ziele umgesetzt worden. Die Benutzung des `ExtendedSoapServer` unterscheidet sich nur durch die Anwendung von XML Parsern. Die WSDL-Beschreibung und die Web Service-Klasse bleiben dagegen völlig identisch.

Die unabhängige Implementierung der Sicherheitsaspekte konnte durch das Stategy-Pattern gelöst werden. Eine Umsetzung von weiteren Standards steht demzufolge nichts entgegen.

Etwas unschön ist dabei, dass der `ExtendedSoapServer` nicht direkt von dem PHP eigenen SoapServer abgeleitet werden konnte. Es ist daher eher eine Frage der Zeit, bis diese Lösung etwas umstrukturiert werden muss. Dies wird aber, wie schon zuvor erwähnt, mit wenigen Handgriffen zu machen sein.

Es kann auf Grund dessen im Gesamten eine positive Bilanz gezogen werden, vor allem dadurch, dass die Interoperabilität gewährleistet ist. (siehe V-H Interoperabilität)

Der Server ist vollständig erstellt. Es fehlt dabei allerdings die Möglichkeit die abgesicherten Web Services unter PHP zu verwenden. Dies soll an dieser Stelle nachgeholt werden.

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

F. Umsetzung des Clients in PHP

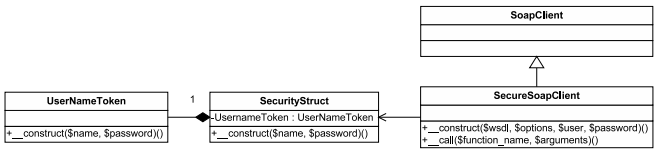

Abb. 9. Klassenstruktur des Clients

Der Client ist punktuell für das Username Token Profile erweitert worden. Dabei konnte hier von einer Vererbung profitiert werden. Aus der von PHP gelieferten Klasse `SoapClient` wird der `SecureSoapClient`. Dazu wurde der Konstruktor leicht verändert, damit die zusätzlichen Parameter wie Name und Passwort entgegengenommen werden können. Dieser bietet demnach nun 4 Parameter an `__construct($wsdl , $options, $user, $pw)` in denen die zusätzlichen Informationen übergeben werden.

Mit den zusätzlichen Daten generiert der Client bei jedem `__call()` Aufruf automatisch den benötigten SoapHeader. Die Benutzung der Klasse ist dabei idealerweise nahezu identisch zum SoapClient und unterscheidet sich lediglich im erweiterten Konstruktor.

Die Erstellung des SoapHeaders ist unter PHP allerdings eine Sache für sich. Es ist erforderlich eine Struktur von Structs zu generieren, um einen konformen Header erzeugen zu können. Hierbei kommen auch die PHP eigenen Klassen `SoapVar` und `SoapHeader` zum Einsatz. Der sogenannte `SecurityStruct`, welcher das Username Token kapselt, wird `SoapVar` übergeben, welcher daraus eine XML konforme Darstellung erzeugt. Das Ergebnis wird anschließend dem `SoapHeader` übergeben. Dabei ist leider zu beachten

das type-Attribute innerhalb eines Tags nicht erzeugt werden können. Damit ist die Interoperabilität vom PHP Client zu beliebigen Web Service Security Servern leider nicht möglich. Auf Grund dessen, dass der Client nur PasswortDigest unterstützt, aber dies im Tag nicht identifizieren kann, wird er von anderen Servern als PasswortText interpretiert und dies sorgt für eine Ablehnung der Anfrage (PasswordText versus PasswortDigest). Bei dem erstellten `ExtendesSoapServer` ist dies kein Problem, da er dies an Hand 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 Großen und Ganzen ist die Implementierung des Clients recht unspektakulär, jedoch werden einige nicht dokumentierte Funktionen in PHP benutzt, was die Entwicklung weitaus schwieriger gestaltete, als anfänglich gedacht wurde.

Dieser Client kann somit durchaus als Vorlage für eigene Security Vorstellungen dienen. Eine Umstellung auf PasswordText wäre relativ problemlos möglich, um Interoperabilität zu anderen Servern gewährleisten zu können.

G. Einbettung in das Gesamtsystem

Da nun die Implementierung des Servers und des Clients abgeschlossen wurde, soll nun ein Blick auf das Gesamtsystem geworfen werden. Dabei wird das altbekannte Bild [5] Abb. 10 zu Grunde gelegt. In diesem wurde nun der Security Agent detaillierter dargestellt. Extended Server übernimmt dabei die Kapselung des eigentlichen PHP Soap Server und leitet die Anfrage an die zwei Parser weiter, die damit entsprechend ihrer Funktion umgehen. Als letzten neuen Bestandteil übernimmt User Token Check die Authetifikation mittels der Datenbank und dem User Management.

Die WSDL Adminoberfläche wurde so erweitert, dass sie mit einem zusätzlichen Flag sichere Web Services erstellen kann.

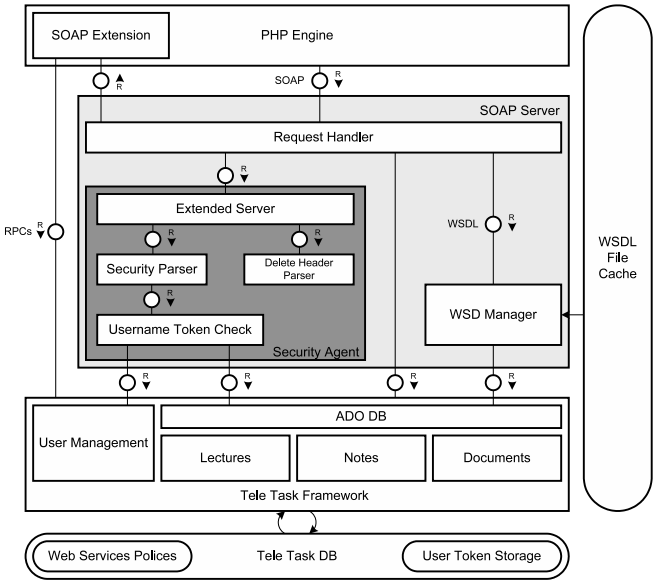

Abb. 10. Gesamtaufbau mit Sicherheit

H. Interoperabilität

Um die Interoperabilität nachzuweisen, wurde der sichere Web Service einmal mit einem C# und zum anderen mit einem Java Client getestet. Beide Interoperabilitätstest verliefen erfolgreich und sind dem Anhang des Projektes zu entnehmen. Die Umsetzung des User Token Profil auf Client Seite wird hier nicht näher erläutert. Es sei hier auf die Referenzimplementierungen[6] verwiesen.

VI. RESTful Web Services

A. Representational State Transfer

Der Begriff REST bzw. Representational State Transfer geht auf R. T. Fielding zurück und wird in [10] im Zusammenhang mit Web Services näher beleuchtet. Für Web Services im Allgemeinen ist eine Implementierung des REST-Archtiekturstils 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, selbst jedoch jeweils komplett unabhängig davon eigene Nachrichtenprotokolle und Schnittstellen definieren.

Wie in [10] dargelegt, hat es verschiedene Vorteile, eine allgemein gültige 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.

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

[6] Microsoft Web Services Enhancements 3.0 www.microsoft.com und Java Axis 1.3 http://ws.apache.org/axis/

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 eine Art *Deployment Descriptor* generiert, welcher vom 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 `RESTServers` 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 klassische Web-Applikation, die der einfachen Konfiguration des Web Service Frameworks dient. Das Policy-PlugIn hingegen fungiert als Filter für Web Service Klassen bzw. deren Methoden.

A. Administration Tool

Das Administrations-Tool soll dem Benutzer die Handhabung des Web Service Frameworks erleichtern. Das Web Service Framework stellt eine Handvoll nützlicher Programme zur Verfügung, die jedoch einzeln bedient werden müssen. Hier setzt das Administrations-Tool an, dass eine intuitive und zentrale Möglichkeit der Administration und Konfiguration bieten soll.

Das Administrations-Tool nutzt eine eigene Bibliothek, die die grundlegende Funktionalität zur Verfügung stellt, in dem sie 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 zu konfigurieren. Klassen werden entweder automatisch in einem gegebenen Verzeichnis gesucht oder können einzeln angegeben werden. Dabei kann schon der Programmierer einer Web Service Klasse mit Hilfe des Tags `@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. 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. 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 dient.

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

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

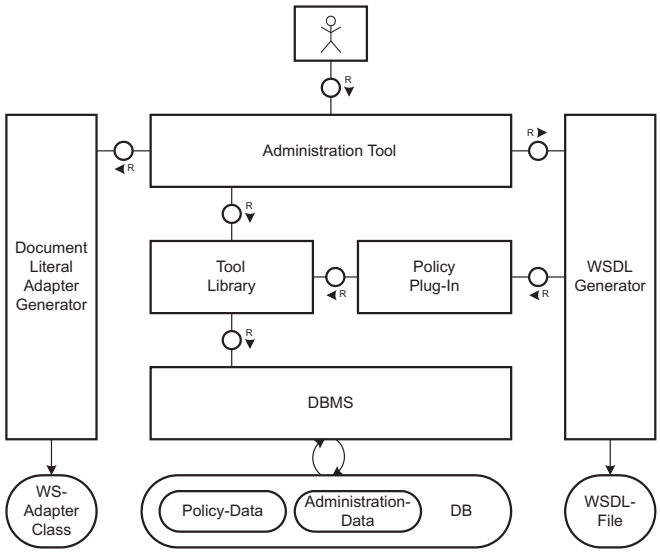

Abb. 11. Aufbaudiagramm des Administrations-Tools

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 Datenbankanbindung, 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.

VIII. Anwendungsbeispiel

Parallel zur Erstellung des Web Service Toolkits wurde eine Beispielanwendung, die auf der alten tele-TASK-Datenbank aufbaut, entwickelt. Die 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 erforderlich. Ansonsten kann der WSDL-Generator die Elemente nicht korrekt identifizieren und es kann keine WSDL-Datei erzeugt werden. Für eine korrekte Implementierung der auf ADOdb basierenden Datenbankabstraktionsschicht, welche mittels einem Web Service angesprochen wird, müssen minimal das Mapping und die Initialisierung durchgeführt werden. Sie bilden den Grundbedarf an Klassen.

A. Aufbau des Beispiels

Es werden je Datenbanktabelle zwei Klassen benötigt. Zum einen müssen die Werte der Tabellenzeilen angesprochen werden können. Dazu werden diese als Objekte behandelt. So lassen sich alle Werte mittels Gettern und Settern 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 ist class.Lecture.php und queryLecture.php ein solches Klassenpärchen, wobei letzteres die Datenbankanfragen zur Verfügung stellt und abarbeitet. Dafür werden die Objekte aus der Anwendungslogik auf die entsprechende Zeile in der Datenbanktabelle gemappt.

B. 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 einer Zeile in der Datenbanktabelle entspricht. Zudem ist je eine Funktion, welche alle Werte auf den Standardwert zurücksetzen kann, eingefügt worden. Den größten Umfang nehmen die Getter und Setter 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 queryXyz.php der benötigte Datenblock übergeben und nach der Anfrage das Ergebnis in erwarteter Form zurückgeliefert. Um dies zu ermöglichen, muss für jede denkbare, sinnvolle Anfrage eine Methode mit der gewünschten Funktionalität vorliegen. In der queryLecture.php ist es unter anderem sinnvoll alle Vorlesungs-Elemente, alle Vorlesungen eines bestimmtes Autors oder eines bestimmten Themas zurückzugeben. Zudem wird eine Methode benötigt, um eine Instanz der Zielklasse zu ermöglichen, damit der Zugriff auf die Datenbank serialisierbar wird. Im gegebenen Beispiel ist die Erzeugung des Rückgabewertes der Übersichtlichkeit halber in einer weiteren Methode gekapselt worden. Dabei werden Rückgabewerte der Datenbank abgefangen und auf die Objekte gemappt, 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. Da die Algorithmen zum Ermitteln der Typinformationen sich nicht direkt im WSDL-Generator befanden, konnten wir gleich eine ganze Familie von Web Service Werkzeugen auf Basis der Extended Reflection API entwickeln. Die Fähigkeiten der Extended Reflection API sind sicherlich für viele weitere PHP Projekte interessant. Der eingeführte Annotation-Mechanismus könnte zudem eine Grundlage für die Nutzung von neuen dynamischen Programmiertechniken wie zum Beispiel aspektorientierter 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 einem 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 Profil*: 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* Addision-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.