Web Services Facade for PHP5

Andreas Meyer, Sebastian Böttner, Stefan Marr
Seminar Konzepte und Methoden der Web-Programmierung 2005
Hasso-Plattner-Institut für Softwaresystemtechnik
{andreas.meyer, sebastian.boettner, stefan.marr}@hpi.uni-potsdam.de

Abstract

Web Services sind inzwischen eine Art "Must-Have"-Feature für jede webbasierte Anwendung, um Informationen zur weiteren Verarbeitung über Systemgrenzen hinweg bereit zu stellen.

Im PHP Umfeld gibt es hierfür momentan noch keine Lösung, die es ermöglicht, mit möglichst wenig Aufwand diese Funktionalität einer bestehenden Anwendung hinzuzufügen.

In diesem Paper wird beschrieben, wie so eine Lösung aussehen könnte, welche Möglichkeiten es im Bereich Authentifikation über SOAP gibt und was für Anforderungen an ein zu erweiterndes System gestellt werden, um den beschriebenen Ansatz verwenden zu können.

Keywords: Web Services, PHP, PHPDoc, WSDL, Security, Username Token Profile

1. Zielsetzung

Für die neue Tele-Task Seite sollen im Rahmen des Seminars Tools geschaffen werden, die es ermöglichen ohne größeren zusätzlichen Aufwand sämtliche Inhalte der Seite per Web Services, genauer gesagt über SOAP, anderen potentiellen Partnerseiten zur Verfügung zu stellen.

Der Königsweg, dies zu realisieren wäre es, vollkommen unabhängig von der letztendlichen Struktur und Implementierung der Seite, diese Services anbieten zu können. Um dies zu erreichen, ist es nötig Tools zu schaffen, welche die Klassen des neu entstehenden Tele-Task Frameworks (TT-Framework) analysieren können und daraus alle notwendige Daten extrahieren um die Beschreibung bzw. Definition der Web Services nach außen zur Verfügung stellen zu können. Weiterhin wird eine Komponente benötigt, welche Anfragen an die Web Services zentral verarbeiten und an das Framework weiterreichen kann. Dieser SOAP-Server soll darüber hinaus auch in der Lage sein, im Bedarfsfall eine Authentifikation des Benutzers zu ermöglichen, um beispielsweise geschützte und personalisierte Web Services anbieten zu können.

Als Grundlage für diese Arbeit dient unter anderem die „Einführung in XML Web Services" [XMLWS]. Die grundlegenden Technologien SOAP und WSDL werden dort näher behandelt.

2. Lösungsansatz

Aus der Situation heraus, dass die neue Tele-Task Plattform parallel entwickelt wird und somit die Web Service-Funktionalitäten nicht einem bestehenden System hinzugefügt werden, von dem bereits alle Aspekte bekannt sind und auch um die somit entstehende Web Service Facade eventuell in anderen Projekten problemlos weiter verwenden zu können, wurde beschlossen als eine Grundvoraussetzung festzulegen, dass für die Nutzung als Web Services keine programmiertechnischen Anforderungen an die potentiellen Service-anbietenden Klassen gestellt werden dürfen.

Dies bedeutet, dass weder der Zwang bestehen soll, ein spezielles Interface zu implementieren oder von einer speziellen Klasse zu erben, noch sonstige konkrete Anforderungen an die Signaturen der Methoden bzw. den Aufbau der Klassen gestellt werden dürfen. Darüber hinaus soll es natürlich auch nicht nötig werden, am Ende von Hand spezielle Klassen implementieren zu müssen, welche die Web Services bereitstellen.

Wenn sich diese Anforderung so realisieren lässt, eröffnet dies theoretisch die Möglichkeit, zu einer beliebigen Anwendung mit Hilfe der hier entwickelten Tools, in einem Arbeitsgang und vor allem idealer Weise ohne Modifikationen am Quellcode, ein SOAP-Interface hinzuzufügen und somit Web Services anbieten zu können.

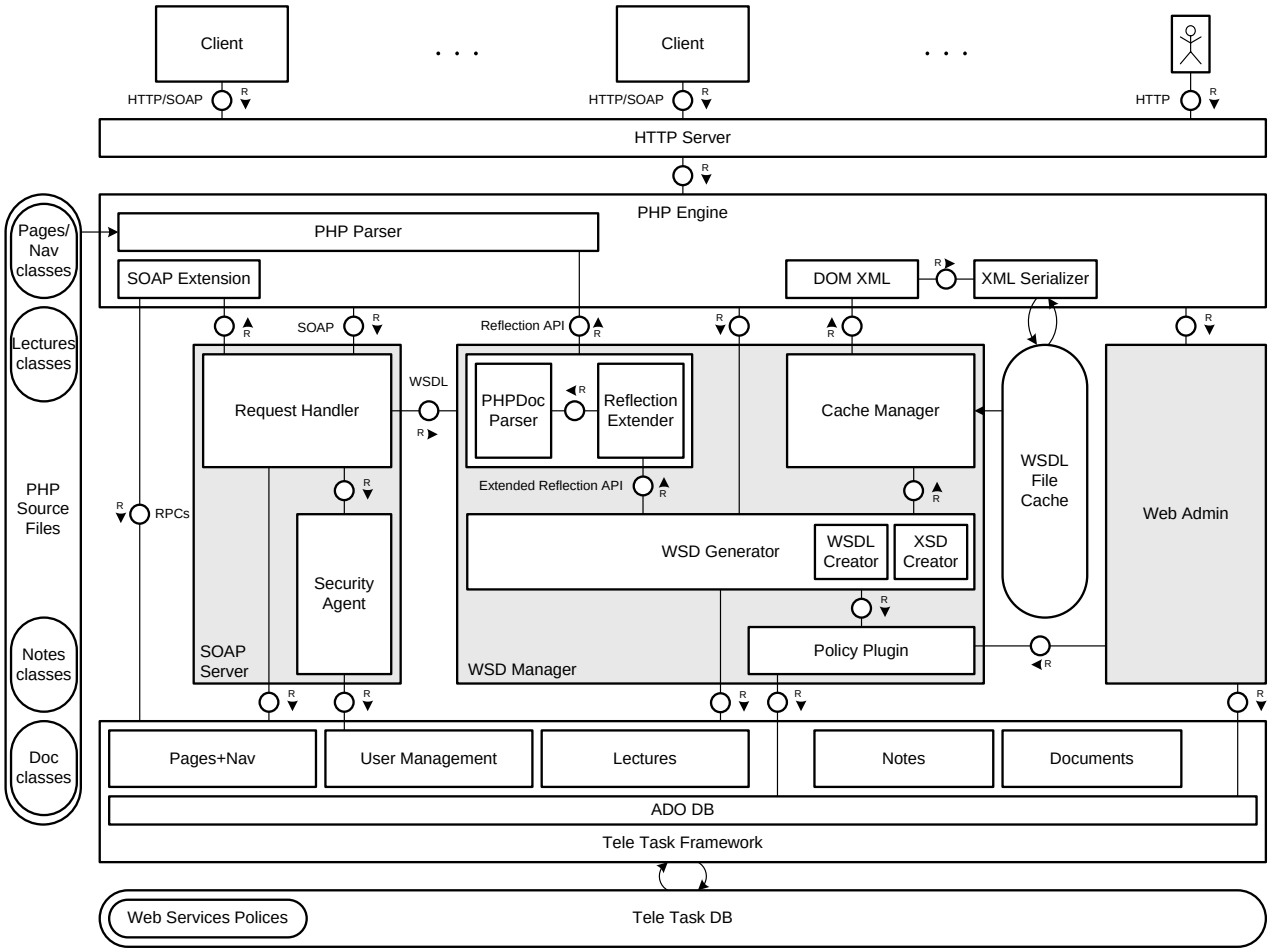

Abbildung 1 Web Service Facade Architektur

2.1 Architekturentwurf

In Abbildung 1 ist der geplante Aufbau des Systems dargestellt. Die zentralen Komponenten sind dabei der SOAP Server sowie der WSD Manager (Web Services Description Manager). Zusätzlich gibt es noch die Web Admin Komponente, die ebenfalls im Rahmen des Praktikums entstehen soll.

In der PHP Engine eingebettet liegen die verschiedenen genutzten Module und Extensions. So bietet der PHP Parser über die Reflection API Zugriff auf die meisten Informationen, die wichtig sind um ein WSDL-File aus einer PHP-Klasse zu generieren. Es werden durch die Reflection API Informationen zu den Klassen, Funktionen und Exceptions bereitgestellt. So ist es zum Beispiel möglich Abzufragen, welche Methoden eine Klasse anbietet, welche Interfaces implementiert werden und was für Parameter erwartet werden. Die DOM-Extension ermöglicht es XML-Dateien nach dem Document Object Model [W3CDOM] als Objektbaum zu erstellen und zu verarbeiten. Die SOAP Extension implementiert die wichtigsten Funktionalitäten, um RPCs über SOAP auf PHP-Klassen und Methoden abzubilden.

Zugegriffen wird auf das System über einen HTTP-Server, welcher die Anfragen von SOAP-Clients oder Web-Browsern an die PHP Engine durchreicht. Für uns sind dabei vor allem SOAP-Anfragen, Anfragen nach den WSDL-Files und Zugriffe auf den Web-Admin über einen Browser von Interesse.

Unter allem liegt nach dieser Vorstellung das TT-Framework, welches auf der einen Seite den Zugriff auf die Datenbank über einen ADOdb-Abstraktionsschicht ermöglicht und besonders wichtig, die Klassen der einzelnen Module enthält, welche die Web Services anbieten sollen. So bietet es sich beispielsweise an, über das Lectures Modul Informationen zu den Vorlesungen nicht nur auf den HTML-Seiten selbst, sondern auch über SOAP verfügbar zu machen.

Für die gewählte Betrachtung des Systems sind nur die persistenten Speicher interessant. Darunter fallen die PHP-Dateien, welche der Parser lesen können muss und auch die WSDL-

Dateien, die schließlich generiert und zur Verfügung gestellt werden müssen. Zusätzlich wird es in der Datenbank noch die Konfigurationsinformationen zu den Web Services geben.

Nun zu den zentralen Komponenten. Das funktionale Herzstück des WSD Managers ist der WSD Generator, welcher über eine erweiterte Reflection API an alle relevanten Informationen zu Klassen, ihren Methoden und Attributen, sowie den Methodensignaturen kommt. Die Erweiterung ist nötig, da PHP als schwach typisierte Sprache selbst nicht genügend Informationen bereitstellt, um einen Web Service komplett anhand einer Klassendefinition beschreiben zu können. Die zusätzlichen Informationen müssen über PHPDoc-Tags innerhalb der Dokumentation zu den einzelnen Elementen notiert werden. Ein Bespiel dazu gibt es im fünften Abschnitt.

Der WSDL- und der XSD-Creator nutzen nun die gewonnen Informationen, die Beschreibung der Web Services sowie der verwendeten Datentypen (XML-Schema Definition [W3CXSD]) zu generieren. Mit Hilfe des Cache Managers wird erreicht, dass nicht mit jedem Abruf der WSDL-Definition zu einem Dienst, diese neu generiert werden muss. Dabei wäre die einfachste Implementierung ein Ablegen von statischen Dateien auf dem Web Server. Eventuell werden noch weitere Mechanismen implementiert, um z.B. bei Aktualisierungen der Web Services Klassen, die dazugehörige Beschreibung automatisch neu zu generieren. Dies hängt jedoch von den letztendlichen Flexibilitätsanforderungen an die Web Service Facade ab. Um darüber hinaus dem Administrator der Plattform eine Steuerungsmöglichkeit zu geben, welche Web Services genau angeboten werden, prüft das Policy Plugin beim Generieren der WSDL-Dateien, welche Einstellungen zu den Klassen hinterlegt wurden und kann so komplette Klassen oder einzelne Methoden von der Veröffentlichung als Web Service ausnehmen.

Über den SOAP-Server werden später alle Anfragen an die angebotenen Web Services laufen. Dieser wird anhand der Anfrage entscheiden, welche Klassen er zum Verarbeiten benötigt und diese entsprechend an die SOAP Extension übergeben sowie die Ausführung anstoßen. Zusätzlich bietet er die Möglichkeit, bei einem Request den entsprechenden Nutzer gegen das TT-Framework zu authentifizieren.

Die letzte Komponente, der Web Admin, erlaubt es, die Einstellungen, auf welche das Policy Plugin für seine Arbeit zurückgreift, zu verwalten. Darüber hinaus bietet der Web Admin die Möglichkeit, die Dokumentation von Methoden, welche in den WSDL-Files hinterlegt wird, vor der Veröffentlichung noch zu modifizieren um sie eventuell zu erweitern, oder Informationen die nur für den internen Gebrauch gedacht sind, auszusparen.

2.2 Projektplanung

Für die Umsetzung wurden die Aufgaben auf drei Teams verteilt. Das Server Team nimmt die Implementierung des SOAP Servers und die Entwicklung einer Beispiel-Implementierung anhand der alten TT-Datenbank vor. Letzteres soll auch als Grundlage für eine eventuelle Fall-Back Lösung dienen können, falls die Ergebnisse des Seminars sich nicht in zufrieden stellendem Maße zu einer Gesamtlösung integrieren lassen. Es gilt also die grundlegenden Operationen auf der TT-DB als Web Services zur Verfügung zu stellen.

Mit den Sicherheitsaspekten beschäftigt sich das so genannte Security Team, hier gehören vor allem das Evaluieren von in diesem Bereich relevanten Standards sowie die anschließende Implementierung der wichtigsten Features zum unterstützen des gewählten Standards, zu den Hauptaufgaben.

Für die restlichen Aufgaben, wurde das Description Generation Team gebildet. Hier steht die Quellcode gestützte Generierung von WSDL-Files im Vordergrund, aber darüber hinaus werden von diesem Team die Web-Oberfläche zur Administration sowie die Dokumentation der Tools zur Verfügung gestellt.

3. Sicherheitsaspekte

3.1 Ziele des Sicherheitskonzeptes

Grundsätzlich soll das Sicherheitskonzept verschiedenen Benutzergruppen mit unterschiedlichen Befugnissen den Zugriff auf das System, d.h. auf die Web Services, ermöglichen.

Dieses Konzept soll von der restlichen Anwendung gekapselt implementiert werden, so dass keine Abhängigkeiten zu implementierten Klassen oder der WSDL-Datei bestehen. Darum sieht die modellhafte Sicht auf den Ablauf der Autorisierung vor, dass es einen Proxy geben

wird, welcher alle Nachrichten eines Auftraggebers abfängt und entscheidet, ob die Anfrage an das System durchzureichen oder mit einer Fehlermeldung zu beantworten ist.

Weiter soll, dem Grundgedanken von Web Services folgend, durch die Authentifizierung bzw. die Autorisation der zustandslose Charakter von Web Services beibehalten werden.

3.2 Vorstellung der verschiedenen Implementierungsvarianten

Letztendlich konkurrieren noch zwei verschiedene Lösungsansätze miteinander – ein „Token Framework“ und das „Username Token Profile 1.0“.

3.2.1 Token Framework

Dieses Verfahren trennt die Authentifizierung von der Autorisierung des Anfragestellers derart, dass zuerst eine Anmeldung am Register Server stattfinden muss und der Anfragesteller nach erfolgreicher Authentifizierung mittels des zurückerhaltenden Tokens Zugriff am Secure Server auf die Web Services erhalten kann.

Für die Authentifizierung wird auf Clientseite der Benutzername und das Passwort abgefragt und mittels einer SOAP-Nachricht an den Register Server übermittelt. Die Übermittlung selbst muss angesichts der Klartextübertragung von SOAP-Nachrichten abgesichert werden. Dazu existieren zwei Alternativen, zum einen über eine HTTPS-Verbindung oder zum anderen mittels eines Secure Socket Layers.

Nach dem Empfangen der SOAP-Nachricht mit den Authentifizierungsdaten überprüft der Register Server die angegebene Kombination auf Korrektheit. Im Falle eines positiven Tests, registriert er eine neue Session und erzeugt die zugehörige Session-ID. Diese wird an den Client übertragen und von selbigem lokal hinterlegt. Des Weiteren initiiert der Register Server das Setzen einer globalen Variable, aus welcher in Abhängigkeit von der Session-ID die Zugriffsrechte ermittelt werden können, im Global User Object. Existiert die Benutzername/Passwort-Kombination nicht, so kann dem Benutzer ein spezieller Account, wie z.B. ein Gastzugang, gewährt und ihm dies mitgeteilt werden. Alternativ wird der Anfrager mittels einer Fehlermeldung auf die falschen Anmeldungsdaten hingewiesen und die Authentifizierung abgelehnt. Die Wahl der Vorgehensweise liegt beim Administrator des Systems. Durch die Nutzung des Session-Mechanismus und der Speicherung der entsprechenden Daten in selbigen geht der zustandslose Charakter der Architektur verloren.

Für die Autorisation wird auf Clientseite nunmehr nur das vom Register Server bekommene Token an jede Anfrage an den Secure Server mit angehängt, d.h. die Session-ID wird im Header der SOAP-Anfrage mit übertragen. Der Secure Server extrahiert die Session-ID aus dem Header und initialisiert damit das Benutzer–Objekt des TT-Frameworks mit den Daten des Anfragestellers. Auf dieses Benutzer-Objekt greifen dann die eigentlichen Service-Methoden im TT-Framework zu und können so einen Fehler ausgeben, falls der Benutzer nicht die Befugnis besitzt die von ihm gewünschte Aktion auszuführen.

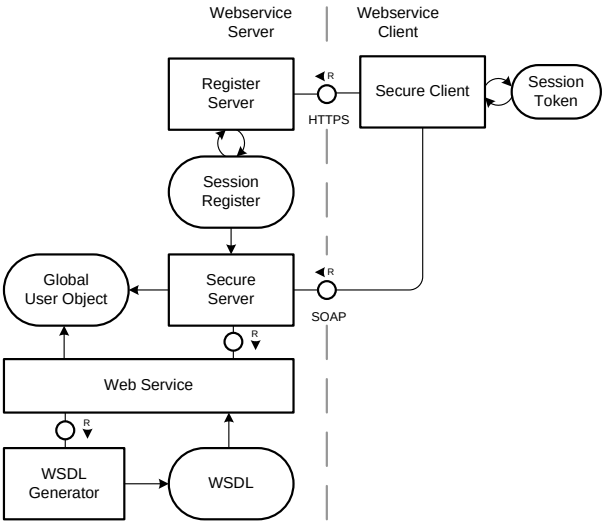

Abbildung 2 Grobstruktur Token Framework

Ein Vorteil dieses Sicherheitskonzeptes ist, dass ausschließlich verbreitete Standards miteinander in Verbindung gebracht werden und das Konzept umschließen. Außerdem ermöglicht die Nutzung von PHP 5 das Einsetzen von zustandsbehafteten Web Services. Dies widerstrebt einem der von uns gesetzten Ziele, so dass die Nutzung von zustandsbehafteten Web Services auch als Nachteil einzustufen ist. Das schwerwiegendere Problem ist allerdings die Klartextübertragung der Authentifizierungsdaten vom Secure Client zum Secure Server. Somit ist eine Verschlüsselung des Passwortes und/oder eine gesicherte Verbindung Voraussetzung für einen Einsatz. Gleiches gilt für die Verbindung zwischen dem Secure Client und dem Register Server, wobei ein Secure Socket Layer auf Grund des erhöhten Overheads hier

eher nicht zum Einsatz kommen wird.

3.2.2 Username Token Profile 1.0

Dieses Verfahren arbeitet ähnlich wie das Token Framework mit dem Unterschied, dass bei jeder Anfrage an einen Web Service der Benutzername und das Passwort übertragen wird.

Prinzipiell ist das Username Token Profile ein Teil des OASIS Web Services Security Standards (WSS) und unterliegt einer stetigen Weiterentwicklung. Der OASIS Standard 1.0 umfasst das SOAP Message Security v1.0, das erwähnte Username Token Profile v1.0 und das X.509 Token Profile. Die beiden letztgenannten Teile verfolgen das gleiche Ziel, die Sicherung von Web Services, auf unterschiedlichen Wegen und setzen jedoch beide auf die SOAP Message Security Spezifikation, welche die Basis zur Verfügung stellt und mit unterschiedlichen Tokenarten zurecht kommt, auf. Der Hauptunterschied zwischen den beiden Profilen liegt im Typ des genutzten Token.

Die Entscheidung über das Zulassen oder Abweisen einer Anfrage beim Username Token Profile 1.0 wird im Security Abschnitt im SOAP-Header getroffen. Dieser Abschnitt ist standardmäßig folgendermaßen definiert:

```
<wsse:Security>
  <wsse:UsernameToken wsu:Id="example">
    <wsse:Username>Example</wsse:Username>
    <wsse:Password Type="..#PasswordDigest">
      weYI3nXd8LjMNVksCKFV8t3rgHh3Rw==
    </wsse:Password>
    <wsse:Nonce>
      WScqanjCEAC4mQoBE07sAQ==
    </wsse:Nonce>
    <wsu:Created>
      2003-07-16T01:24:32Z
    </wsu:Created>
  </wsse:UsernameToken>
</wsse:Security>
```

Beispiel-Security-Abschnitt

Der Abschnitt besteht aus zwei notwendigen (`Username` und `Password`) und zwei optionalen (`Nonce` und `Created`) Tags. Im Username-Tag wird der Benutzername des Anwenders hinterlegt. Im Password-Tag kann man bei Bedarf eine Typangabe hinzufügen, wobei die Auswahl zwischen `PasswordText` und `PasswordDigest` besteht. Ersteres ist der Standardwert und wird automatisch angenommen sofern keine Typisierung vorgenommen wurde. Bei der Wahl dieses Typs steht das Passwort in einer beliebigen Form in diesem Tag – Klartext, kodiert, als Hashwert usw. sind möglich. Durch die Wahl von `PasswordDigest` als Typ ist die Freiheit bezüglich der Passwortkodierung eingeschränkt. Selbiges wird immer als SHA-1-Hashwert und Base64-kodiert im Tag abgelegt. Im Nonce-Tag wird ein zufällig generierter String hinterlegt. Dieser ist standardmäßig Base64-kodiert, kann aber durch Angabe eines Typattributs, wie es in `<wsse:BinarySecurityToken>` definiert ist, geändert werden. Der Created-Tag beinhaltet abschließend einen Zeitstempel zur Angabe des Erstellungszeitpunktes.

Entscheidet man sich im Password-Tag für den Typ `PasswordDigest` und ist wenigstens eines der beiden optionalen Tags angegeben, so muss der im Password-Tag hinterlegte Wert folgendermaßen aufgebaut sein:

```
PasswordDigest =
  Base64(SHA-1(nonce + created + password))
```

Die Wahl dieser Reihenfolge hat den Vorteil, dass das geteilte Geheimnis am Ende der Eingabe steht und es damit möglichen Angreifern erschwert wird, unrechtmäßigen Zugriff auf das System zu erhalten. Wäre die Reihenfolge anders gewählt, so könnte ein Angreifer die Eingabe derart mit einem Nonce oder einem Zeitstempel erweitern, dass er sich mittels Erzeugung eines neuen Hashwertes Zugriff auf das System verschafft. Wenn das Passwort dahingehend am Ende der Eingabe steht, kann die Eingabe nicht beliebig verlängert werden, da sie in jedem Fall mit dem für den Angreifer unbekannten Passwort abschließen muss.

Es gibt seitens der Entwickler des Standards, die folgenden drei sicherheitsrelevanten Empfehlungen, um das Problem der Replay-Attacken anzugehen:

1. Jedes UsernameToken, welchem entweder das Nonce oder der Zeitstempel fehlt, sollte abgewiesen werden.
2. Der Zeitstempel sollte nur eine begrenzte Zeit gültig sein und jedes UsernameToken mit einem abgelaufenen Zeitstempel sollte abgewiesen werden.
3. Die Nonces sollten serverseitig für einen gewissen Zeitraum, der die Gültigkeit des Zeitstempels überschreitet, mitgecached und jedes UsernameToken mit einem sich wiederholenden Nonce sollte abgewiesen werden.

Ein großer Vorteil dieses Sicherheitskonzeptes ist es, dass es ein offener, in der stetigen Weiterentwicklung befindlicher Standard ist und große Firmen wie IBM und SUN (java) auf ihn zurück-

greifen. Des Weiteren kann er unabhängig von der Programmiersprache in vielen Bereichen implementiert werden, so dass unterschiedlichste Clients auf diesem Konzept aufsetzen können. Zu diesem Konzept gibt es nur proprietäre oder nicht auf allgemeinen Standards basierende Lösungen als Alternativen. Über die Problematik der sicheren Passwortübertragung sollte man sich dennoch weitergehende Gedanken machen, da der Standard nicht vorschreibt, dass das Passwort verschlüsselt übertragen werden muss. Eine Möglichkeit die ungesicherte Übertragung des Passwortes zu umgehen, wäre einen sicheren Kanal zwischen dem Client und dem Server zu nutzen und auf dem Server einen Hashwert des Passwortes zu erstellen und diesen mit dem in der Datenbank hinterlegten Hashwert zu vergleichen. Der Standard sieht die Möglichkeit vor, dass sowohl Passwörter selbst, als auch Passwortäquivalente verwendet werden können. Dies setzt nur voraus dass beide Seiten das gleiche Wissen besitzen. Ein weiteres Problem stellt das PasswordDigest dar. Dieses ist für eine gewisse Zeit auf jeden Fall gültig und ermöglicht in diesem Zeitraum den Zugriff auf das System, sobald ein Angreifer in den Besitz des PasswordDigest gelangt. An selbiges kann er beispielsweise durch einen man-in-the-middle-Angriff gelangen. Ein Abfangen der Anfrage des Clients an den Server, bevor die Anfrage den Server erreicht, ermöglicht dem Angreifer mindestens einmal Zugriff auf das System. Durch den Einsatz dieser dritten Empfehlung, d.h. cachen der Nonces, kann das Problem bezüglich der Gültigkeitsdauer des PasswordDigest aber minimiert werden und das Risiko wäre auf genau einen möglichen missbräuchlichen Zugriff pro abgefangenem, noch nicht verwendeten PasswordDigest reduziert.

3.3 Implementierungsentscheidung

Die Entscheidung ist zugunsten des Username Token Profiles in der Version 1.0 gefallen. Dies begründet sich vor Allem darin, dass das Schaffen von persistenten Web Services der einzige echte Grund für die Wahl des Token Frameworks wäre und diese aber ausgeschlossen wurden. Weitere schwerwiegende Nachteile des Token Frameworks sind der proprietäre Charakter und der nicht einschätzbare Aufwand für die Implementierung.

4. Funktionale Aspekte von Quellcode-Dokumentation

4.1 Allgemein

Dokumentation in Software-Programmen hat zumeist eine beschreibende Funktion und soll dem Leser Informationen über die Funktionsweise und die Verwendung des kommentierten Code-Abschnittes bereitstellen, sie kann jedoch auch eine funktionelle Verwendung finden, wenn die in der Dokumentation enthaltenen Informationen wesentliche Charakteristika des Code enthalten.

Die maschinelle Verarbeitung dieser Dokumentation setzt jedoch eine einheitliche und strukturierte Art der Kommentierung voraus. Die Standardisierung der Codegestaltung wirkt sich sowohl auf den beschreibenden als auch funktionellen Charakter der Dokumentation aus.
Einheitliche Kodierungsvorschriften erhöhen daher nicht nur die Lesbarkeit und Wartbarkeit von Quellcode, die strukturierten Kommentare können darüber hinaus von Parser-Programmen verarbeitet und aufbereitet werden, um eine von den Quelldateien unabhängige Dokumentation zu erstellen, wie dies z.B. in der Java-Welt mit Hilfe des Javadoc-Tools üblich ist.

4.2 PHPDocumentor

Der phpDocumentor ist ein Standard-OpenSource-Tool für PHP, das automatisch Dokumentationen aus dem Quellcode generieren kann. Der Documentor ist dabei selber in PHP geschrieben und kann als Kommandozeilentool oder als einfach bedienbare Web-Oberfläche verwendet werden.

Das eigentliche Ziel des phpDocumentors ist die Aufbereitung der im Quellcode enthaltenen Dokumentation. Denn auch wenn der Quellcode ausreichend und formgerecht kommentiert wurde, bleibt es beim Nutzer die entsprechenden Informationen zwischen den Codefragmenten ausfindig zu machen. Um ein einfaches und für den Menschen leicht verständliches Format zu erhalten, speichert der phpDocumentor die gesammelten Informationen in eines von 15 vorgegebenen HTML-Formaten, in ein PDF-Dokument, ein Windows Helpfile oder eine DocBlock XML Datei. Darüber hinaus ist es

möglich sich eigene Smarty-basierte[1] HTML-Templates zu erstellen um die Dokumentation dem eigenen Stil oder Vorlieben anzupassen.

Der Documentor geht dabei über eine Konvertierung der Quellcode-Kommentare hinaus und fügt selbstständig Links zu Abhängigkeiten oder per `@see` angegebenen Elementen ein. Zusätzlich werden automatisch Vererbungsdiagramme zu allen Klassen des Projektes und TODO Listen aus den entsprechenden Tags erstellt. Dadurch können vollständige Dokumentationen zu Projekten erstellt werden, die abhängig vom Zielpublikum mit Hilfe des Tags `@internal` unterschiedlichen Umfang haben können.

Im Anhang sind weitere Beispiele zur Verwendung zu finden und unter [PHPDOC] ist eine vollständige Liste und Beschreibung der verschiedenen Tags hinterlegt.

5. Gestaltungsrichtlinien im Tele-Task-Projekt

Für das Tele-Task-Projekt wurden die gegebenen Richtlinien zur Gestaltung des Quellcodes überarbeitet und einige Anpassungen vorgenommen, um dem erweiterten Informationsbedarf zur Generierung einer Web Services-Beschreibung gerecht zu werden. (siehe Anhang).

Der Vorteil dieser Richtlinien liegt in der Strukturierung der dargebotenen Informationen. Jeder Kommentar in der Datei liegt in Form so genannter DocBlock-Kommentare vor, die in Beschreibungen und vordefinierte Tags unterteilt sind. Im Gegensatz zu textuellen Beschreibungen lassen sich diese Tags sehr gut zur funktionellen Dokumentation nutzen, da sie bereits kategorisiert sind und sich durch feste Strukturen zur Verarbeitung eignen.

```
//=========================================
/**
* constructor-method of class dbNews
*
* This is the constructor used to instantiate
* dbNews objects
*
* @param integer $id
* @param array<string,string> $data
*/
public function __construct($id,$data=null) {
  $this->_tableName = news;
  $this->_primaryKey = 'id';
  parent::__construct($id, $data);
}
```

Beispiel Quellcode

[1] Smarty ist eine Template Engine für PHP, siehe [SMARTY]

Diese Kommentarblöcke werden für jede Datei, Klasse, Methode oder Variable erstellt und ermöglichen daher eine sehr detaillierte Dokumentation aller Features des Projektes.

Der größte Vorteil des Standards besteht darin, dass bereits vorhandene Parser, wie zum Beispiel der phpDocumentor zur Verfügung stehen, die die Bedürfnisse des TT-Projektes mehr als erfüllen.

5.1 Benötigte funktionale Erweiterungen der Dokumentation

Auch wenn eine detaillierte Dokumentation für das TT-Projekt letztendlich sehr wichtig ist, sind die Informationen, die aus den Tags gewonnen werden können, funktionsentscheidend. Besonders interessant sind hier die Tags `@var`, `@param` und `@return`.

Um die Dienste des TT-Frameworkes als Web Service-Anwendungen bereitstellen zu können, werden WSDL-Dateien benötigt. Diese sollen automatisch von einem WSDL-Generator erzeugt werden und definieren die Kommunikationsschnittstelle zwischen dem Client und dem Server durch Angabe der verfügbaren Funktionsnamen als auch der entsprechenden Funktionsparameter und Rückgabewerte.

Diese Informationen werden durch Quellcode-verarbeitende Programme, also in diesem Fall dem PHP-Parser der Script-Engine, über die ab PHP5 angebotene Reflection API zur Verfügung gestellt. Bei schwach typisierten Sprachen wie PHP ist jedoch die Ermittlung der Typ-Informationen nur bedingt möglich. Um die für die WSDL-Dateien relevanten Daten dennoch automatisch ermitteln zu können und sich aufwendige manuelle Arbeiten zu ersparen, bietet sich hier die bereits angesprochene funktionelle Erweiterung von Kommentaren innerhalb der entsprechenden Dateien an. Im Falle der [STYLE] Richtlinien werden eben jene Daten in den oben genannten Tags dargestellt.

Der WSDL-Generator erzeugt nun, mit Hilfe der erweiterten Reflection API und einem minimalen Parser für die wichtigsten Tags, die für den Web Service nötigen WSDL Dateien. Diese werden automatisch beim Aufruf des Web Services verwendet, um den Server zu konfigurieren. Inkorrekte oder fehlende Typinformationen in der Dokumentation resultieren letztendlich in einer fehlerhaften WSDL-Datei und der Web Service wird damit nicht oder nur fehlerhaft verwendbar

sein. Er kann nur durch langwierige manuelle Erstellung der WSDL Datei verfügbar gemacht werden, daher ist eine strikte und korrekte Einhaltung dieser Regeln enorm wichtig, wenn weitere Web Services dem System hinzugefügt werden sollen. Die vollständigen Richtlinien sind [STYLE] zu entnehmen.

Quellen

[XMLWS05] Einführung in XML Web Services, C. Hartmann, M. Sprengel, M. Perscheid, G. Gabrysiak, F. Menge, 2005

[UTP10] Web Services Security - UsernameToken Profile 1.0, OASIS Standard 200401, March 2004 http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

[WSS11] Web Services Security: SOAP Message Security 1.1, Working Draft - 07 November 2005 http://www.oasis-open.org/committees/download.php/15251/oasis-wss-soap-message-security-1.1.pdf

[W3CDOM] Document Object Model http://www.w3.org/DOM/

[W3CXSD] XML Schema http://www.w3.org/TR/xmlschema-0/

[PHPMAN] PHP.net Manual http://www.php.net/manual/en/ref.soap.php http://www.php.net/manual/en/language.oop5.reflection.php

[PEAR] PEAR Coding Standards http://pear.php.net/manual/en/standards.php

[PHPDOC] phpDocumentor tags - How to use tags in DocBlocks http://manual.phpdoc.org/HTMLSmartyConverter/HandS/phpDocumentor/tutorial_tags.pkg.html

[JAVADOC] How to Write Doc Comments for the Javadoc Tool http://java.sun.com/j2se/javadoc/writingdoccomments/

[SMARTY] Smarty Template Engine http://smarty.php.net/

[ADODB] ADOdb Database Abstraction Library for PHP http://adodb.sourceforge.net/

[STYLE] Style Guide für das TT-Projekt zu finden im Anhang A

[TELETASK] http://www.tele-task.de/

Anhang A: Style Guide

Richtlinien zur Quellcode-Gestaltung

Diese Richtlinien dienen als Grundlage zur automatisierten Erstellung von Web Services aus PHP-Klassen. Darüber hinaus ist ein einheitliches Code-Bild natürlich auch anzustreben, um die Wartbarkeit und Lesbarkeit der Quelltexte im gesamten Projekt zu erhöhen.

1. Allgemeines

Die Länge einer Zeile sollte nicht mehr als 77 Zeichen betragen.
Einrückungen: 2 Leerzeichen pro Ebene, keine Tabs

Als Sprache für Kommentare und Bezeichner ist Englisch zu verwenden.
Bezeichner sind weiterhin in der CamelCase Schreibweise zu notieren wobei Unterstriche (_) zu vermeiden sind.

2. Dateikopf

```
//****************************************************************************
//****************************************************************************
//**                                                                        **
//** ClassName - Kurze Beschreibung dieser Klasse                           **
//**                                                                        **
//** Project: Name des Projekts                                             **
//**                                                                        **
//** @package    Paketname                                                   **
//** @author     Max Mustermann <mail@example.org>                          **
//** @copyright  2005 ....                                                  **
//** @license    URL zur Lizenz   Name der Lizenz                           **
//** @lastchange YYYY-MM-DD - Details zur Änderung                          **
//**                                                                        **
//****************************************************************************
//****************************************************************************
```

3. PHPDoc Kommentare

Ein Dokumentationskommentar wird in HTML geschrieben und gehört vor Klassen, Attribute, Konstruktoren und Methoden. Er besteht aus zwei Teilen, einer Beschreibung und darauf folgenden PHPDoc-Tags, wie zum Beispiel @param, @return und @var.

Die erste Zeile beginnt dabei mit /** um den Anfang zu kennzeichnen.
Der erste Satz sollte eine kurze Zusammenfassung sein, darauf kann durch eine Leerzeile getrennt eine ausführlichere Beschreibung folgen. Wieder durch eine Leerzeile getrennt, können anschließend weitere Angaben mit Hilfe der PHPDoc-Tags hinzugefügt werden. Die letzte Zeile endet mit einem einfachen */ um den Kommentarblock zu schließen.

4. PHPDoc Tags

- **@author** authorname <authoremail@example.com>
  Autor des mit diesem Tag versehenden Elements
- **@copyright** info string
  zur Angabe von Copyright-Hinweisen
- **@deprecated** [info string]
  im optionalen info string können Angaben bezüglich Version und Gründen hinterlegt werden

- **@global** datatype $globalvariablename description
  zum dokumentieren von benutzten globalen Variablen
- **@internal** info string
  für Informationen die nur zum internen Gebrauch bestimmt sind
- **@license** URL name of licence
  zur Angabe der Lizenz unter der dieses Code-Fragment veröffentlich wurde
- **@package** name
  zum Dokumentieren von strukturellen Zusammenhängen
- **@param** datatype $paramname description
  datatype kann einer der folgenden PHP-Typen sein
  - boolean
  - integer
  - double
  - string
  - beliebige Klasse

  Außerdem ist es möglich Arrays von den PHP-Typen anzugeben. Dies geschieht über ein anhängen von eckigen Klammern. Bsp.: integer[], string[][]
  Assoziative Arrays werden wie folgt notiert: array<datatype,datatype> wobei diese Notation nur verwendet werden sollte, wenn die Schlüssel wirklich relevant sind, andernfalls ist die Notation mit eckigen Klammern vorzuziehen. Bsp.:
  array<integer,string> oder array<string,MyClass>

- **@return** datatype description
  zur Angabe des Rückgabewertes einer Methode oder Funktion
- **@see** element
  wird genutzt um auf andere Elemente zu verweisen, z.B. Funktionen, Dateien, Variable oder Klassen
- **@since** info string
  gibt an ab welcher Version dieses Element vorhanden ist
- **@todo**
  zur Dokumentation von beabsichtigten Änderungen
- **@uses** element
  zur Dokumentation von Nutzungs-Beziehungen, z.B. von Konstanten oder Funktionen
- **@var** datatype description
  zur Dokumentation des Typs eines Attributs
- **@version**
  zur Dokumentation der Version des betreffenden Elements

5. Klassen

Jeder Klasse wird eine abtrennende Zeile mit dem Klassennamen, sowie einleitende Dokumentation im PHPDoc Stil vorangestellt.

```
//***** ClassName ********************************************************
/**
* Kurze Beschreibung dieser Klasse
*
* @package    Paketname
* @author     Max Mustermann <mail@example.org>
* @copyright  2005 ....
* @license    URL zur Lizenz   Name der Lizenz
*/
```

6. Attribute

Die Attribute die in einer Klasse definiert werden, sind jeweils einzeln mit einem PHPDoc Kommentar zu versehen. Dabei muss mindestens der Datentyp des Attributes über den **@var**-Tag angegeben werden.

```
/**
* @var string
*/
protected $name;

/**
* @var MyObject[]
*/
private $objects = null;
```

7. Methoden und Funktionen

Methoden und Funktionen müssen mit einer Linie sowie einem einleitenden PHPDoc Kommentar versehen werden.

Dabei sind **@param** und **@return** Pflichtangaben um für die Generierung der WebServices-Beschreibung Informationen über die in der Signatur verwendeten Datentypen zu bekommen.

Wenn die Funktion mehr als 15 Zeilen umfasst, muss hinter der abschließenden Klammer der Funktion ein `//end of functionname`-Kommentar angegeben werden.

```
//========================================================================
/**
* Kurze Beschreibung
*
* Mehrzeilige, detailierte Beschreibung.
* Hier können auch HTML-Tags zum Formatieren genutzt werden.
* Darüber hinaus sind Kommentare in Englisch anzugeben.
*
* @param string $a
* @param integer $b
* @param MyClass $c
* @return string[]
*/
public function getMethod($a, $b, $c = null) {
  return array('a', 'aa', 'b');
}
```

8. Kontrollstrukturen

Bei Kontrollstrukturen, die über mehr als 15 Zeilen gehen, ist nach der schließenden Klammer ein jeweils passender Endkommentar anzugeben, um die Lesbarkeit von langen Code-Abschnitten zu erhöhen. Wenn die Code-Abschnitte nur kurz sind, müssen diese abschließenden Kommentare nicht angegeben werden.

```
//========================================================================
/**
* shows how to use brackets, whitespaces and indentations
*/
public function styleExample() {
  if ($a == $b || !($c > $d + $e)) {
    //do something
```

```
  } //end if
  elseif ($f) {
    //do something
  } //end elseif
  else {
    //do something
  } //end else

  for|foreach|while (...) {
    //do
  } //end for|foreach|while

  do {

  } while ($g);

  switch ($h) {
    case ...:
      ...;
      break;
    default:
      ...;
  } //end switch
} //end of styleExample
```