Wie in der Einleitung beschrieben, stellt die zugrundeliegende Web-Anwendung keine Schnittstellen für den Zugriff auf die vorhandenen Daten zur Verfügung, so dass ein entsprechender Web-Service implementiert werden soll. In diesem Abschnitt sollen die Grundlagen zum Thema Web Service erläutert werden.

* GET

Die Methode GET stellt den lesenden Zugriff auf Ressourcen zur Verfügung. Die Methode verhält sich dabei idempotent und ist sicher bzw. gefahrlos. Idempotent bedeutet in diesem Fall, dass ein lesender Zugriff die gewünschte Ressource unverändert lässt und somit jeder weitere Zugriff auf die Ressource das gleiche Ergebnis liefert. Sicher im Zusammenhang mit der Methode GET bedeutet, dass der Aufruf einer Ressource durch einen Benutzer keinen Seiteneffekt hervorruft, d.h. dass die Anfrage keinen vom Benutzer erwarteten Einfluss auf andere Ressourcen oder den Betrieb des Servers haben darf. [W3C][Bur10]

* PUT

Die Methode PUT ist eine schreibende Methode. Bei der Anfrage an den Server, wird die in der http-Nachricht übertragen Ressource an der eindeutigen Adresse (URI) gespeichert, die in den Meta-Daten des Paketes angegeben ist. Sollte die angegebene URI nicht vorhanden sein, wird sie durch den Server angelegt. Die Methode PUT ist idempotent, d.h. dass die mehrmalige speichernde Anfrage der gleichen Ressource dasselbe Ergebnis herbeiführt. [Bur10]

* DELETE

Die Methode DELETE wird benutzt um Ressourcen auf dem Server zu löschen. Die Methode ist ebenfalls idempotent, da das mehrmalige Löschen einer Ressource das gleiche Ergebnis liefert. [Bur10]

* POST

Die Methode POST ist sehr flexibel einsetzbar. So ist es möglich in den entsprechenden Anfragen bzw. Antworten Daten zu senden oder zu empfangen. Weiterhin können mit der Methode auch die Funktionalitäten von anderen Methoden wie GET und PUT realisiert werden. Allerdings ist PUT weder idempotent noch sicher, weil die in den Anfragen übertragenen Meta-Daten bzw. Parameter ständig veränderbar sind. Zudem bietet PUT weitere Funktionen an, wie das Annotieren von bekannten Ressourcen. [Bur10] [W3C]

* HEAD

Die Methode HEAD stellt die gleiche Funktionalität wie die Methode GET zur Verfügung. Der Unterschied besteht darin, dass der Server bei einer HEAD-Anfrage nicht zwingende den Inhalt der aufgerufenen Ressource zurücksendet, sondern nur die Meta-Daten die die Ressource betreffen.[W3C]

* OPTIONS

Die Methode OPTIONS ermöglicht den Aufruf von Optionen für die Kommunikation zwischen Server und Nutzer und deren Auswertung, ohne die Ressource selbst anzufordern. [Bur10]

* TRACE

Die Methode TRACE stellt Funktionalitäten zur Verfügung mit denen der Nutzer zu Testzwecken Anfragen an den Server schickt und der Server die verarbeitet Anfrage zurücksendet. Mit dieser Methode kann der Nutzer die Rückantwort auswerten und die gewonnenen Informationen zur Fehlerbehebung oder Diagnose nutzen. [W3C]

* CONNECT

Die letzte zu betrachtende Methode ist CONNECT. Diese Methode steht für die Nutzung von Proxy-Eigenschaften zur Verfügung, um dynamisch auf gesicherten bzw. getunnelten Netzwerkverkehr umzuschalten. [W3C]

@Path

Definiert den Pfad mit der die zur Verfügung gestellte Klasse aufgerufen werden kann und bildet so den Knotenpunkt („root resource“) für die mit @Path annotierten Methoden in der Klasse.

@Path(”/test”)

public class TestKlasse{

@Path(/methode1)

public String testMethode1(){

…

}

}

Die im Web-Service aufzurufende URI wäre dann

*http://server:port/webprojekt/test/method*

Für die Einbindung von Meta-Daten der aufgerufenen Methoden gibt es weitere Annotationen:

@GET

Bindet die GET-Methode an die in der HTTP-Anfrage definierte Methode. Diese Annotation ist auch für die HTTP-Methoden PUT, POST, DELETE und HEAD möglich.

@Produces

Über diese Annotation wird der erwartete Rückgabetyp der aufgerufenen Methode festgelegt.

@PathParam

Die Annotation @PathParam dient dazu, die Eingabeparameter einer Methode mit den Werten aus der HTTP-Anfrage zu initialisieren. Der Parameter wird dann in der @Path Annotation der Methode mit geschweiften Klammern angegeben.

Beispiel:

@Path(“/methode2/{parameter}”)

public String testMethode2(@PathParam(“parameter”) String eingabe){

…

}

@FormParam

Die Annotation @FormParam bietet die Möglichkeit Parameter in Form von Name-Werte-Paaren in einer HTTP-Anfrage zu übertragen. Das Einfügen der möglichen Parameter in die HTTP-Anfrage liegt in Verantwortung des Clients.

@Path(“/methode3/”)

public String testMethode3(@FormParam(“parameter”) String eingabe){

…

}

### Unit-Test

Wie bereits beschrieben stellen Unit-Tests Funktionen bereit, um Module eines Programmes, ohne die Einwirkung anderer Module zu testen. Bei der Nutzung von Unit-Tests sollen einige Grundsätze beachtet werden, die nachfolgend kurz aufgezeigt werden sollen.

* Automatisch

Der Aufruf der Testklassen und die Auswertung der Ergebnisse soll automatisch erfolgen. Der Hintergrund dafür besteht in der Wiederverwendbarkeit der Tests, die sofort aufzeigen sollen falls ein Test fehlschlägt. Der manuelle Aufruf birgt die Gefahr, dass Tests vergessen werden bzw. die Auswertung nicht erfolgt. Zudem sollten Tests verwendet werden, die ohne die manuelle Eingabe von Testparametern auskommt, um den automatischen Ablauf zu gewährleisten. Um den automatischen Ablauf weiterhin zu gewährleisten sollen für die Tests Mock-Objekte und isolierte Ablaufumgebung (siehe2.3.4.1)verwendet werden.

* Sorgfalt

Die zu implementierende Testklassen sollen alle möglichen auftretenden Fehlerfälle durchlaufen und eine möglichst große Code-Abdeckung realisieren. Bei großen Software-Projekten kann das Testen der gesamten Funktionalitäten langwierig und unübersichtlich werden. Abhängig von den gewünschten Projektzielen, kann es ausreichen die Tests auf bestimmte Szenarien (falsche Eingaben, Null-Pointer, falsche Datentypen und Rückgabenwerte, o.ä.) zu beschränken.

* Wiederholbar

Die durchzuführenden Tests müssen zu jedem Zeitpunkt, in jeder Reihenfolge in der sie ablaufen, die gleichen Ergebnisse liefern. Für diesen Ablauf ist die Unabhängigkeit der Testklassen untereinander erforderlich. In größeren Projekten, in denen mehrere Entwickler tätig sind, ist zu beachten, dass die Tests der einzelnen Entwickler sich gegenseitig nicht beeinträchtigen dürfen.

* Unabhängigkeit

Testklassen dürfen keine Abhängigkeiten zuvor ablaufender Tests besitzen. Um diesen Aspekt gewährleisten zu können, sollen die setUp() und tearDown()-Methoden (siehe 2.3.3)genutzt werden, um die Testumgebung nach jeder Testklasse zurückzusetzen.

* Professionell

Die Implementierung der Testklassen ist mit gleicher Präzision durchzuführen, wie bei dem Produkt-Code der getestet werden soll. Die Testklassen sind den zu testenden Methoden entsprechend anzupassen, so dass die Funktionalitäten methodenspezifisch überprüft werden.

durch die Entwicklungsumgebung generierten Persistenz-Klassen befinden sich im gleichnamigen „persistence“-Paket und repräsentieren die Abbildung der relationalen Tabellen der zugrundeliegenden Datenbank. Das Paket „helper“ beinhaltete die Klasse „Connection“, die als Hilfsklasse für das Auffinden von Netzwerk-Verbindungen, bestehend aus mehreren Netzwerk-Kabeln implementiert wurde. Die Idee besteht darin, dass ein Netzwerk-Kabel vom Typ „Cable“ mit höchstens zwei „CableInterfaces“ verbunden sein kann. Die „CableInterfaces“ können über das Attribut „objectId“ eindeutig dem dazugehörigen „SapObject“ zugordnet werden. Die Klasse „ConnectionManager“ stellt über das Interface „ConnectionManagerLocal“ die Funktionen bereit, um Netzwerk-Verbindungen zwischen zwei Geräte ausfindig zu machen. Ergebnis dieser Suche ist eine Liste, in der jedes Element ein Objekt vom Typ „Connection“ ist.

Neben dem Interface „ConnectionManagerLocal“ und dessen implementierende Klasse „ConnectionManager“ stehen weitere Klassen und Interfaces für den Zugriff auf die Persistenz-Objekte zur Verfügung. Die Klassen „ObjectManager“, „InterfaceManager“ und „CableManager“ implementieren, die namentlich mit „Local“ erweiterten Interfaces. Die Einbindung der Persistenz-Objekte erfolgt dabei über die „EntityManager“, die durch die „Java Persistence API“ bereitgestellt werden. Die passenden Interfaces stellen anderen Klassen die Methoden bereit, um die gewünschten Objekte anhand bestimmter Suchkriterien aufzurufen. Neben dem lesenden Zugriff stellen die Interfaces Methoden bereit für die Anwendung der sogenannten CRUD-Methoden (siehe 2.1) bereit. Aufgrund der Einschränkungen in diesem Projekt, wurde auf diese Implementierung verzichtet.

Zusätzlich befindet sich in dem Paket „manager“ die Klasse

Die Klasse „AndroidService“ stellt den zentralen Einstiegspunkt für externe Anwendungen auf den Web-Service und die implementierten Methoden bereit. Die entsprechenden Methoden werden mit Hilfe entsprechender Annotationen bereitgestellt. Die benötigten Annotationen werden durch das Framework Jersey zur Verfügung gestellt. Neben den Methoden zur Realisierung der gewünschten Use-Cases werden durch die Klasse „AndroidService“ auch private Methoden für die Umformung der Datensätze in, die zur Übertragung via HTTP benötigten, JSON-Objekte implementiert.