Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Fachstudie Nr. 170

Vergleich von Frameworks zur Implementierung von REST basierten Anwendungen

Markus Fischer und Kalman Kepes und Alexander Wassiljew

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Leymann
Betreuer/in:	Dipl.-Inf. Florian Haupt
Beginn am:	2012-12-17
Beendet am:	2013-06-18
CR-Nummer:	K 6.1, D 2.1, D 2.2, D 2.13, D 3.3

Kurzfassung

Mit steigender Popularität des Architekturstils *Representational State Transfer* veröffentlicht in der Dissertation *Architectural Styles and the Design of Network-based Software Architectures* von Roy T. Fielding im Jahre 2000, wurden und werden immer mehr Arbeiten veröffentlicht die sich mit dem Thema beschäftigen. Darunter befinden sich auch Frameworks und Spezifikationen, die das Entwickeln von *RESTful Services* anhand der Aspekte von REST, wie beispielsweise *Uniform Interface* erleichtern wollen. In dieser Fachstudie werden eine Reihe von Frameworks, für das Entwickeln von *RESTful Services* vorgestellt und anhand eines Kriterienkatalogs ausgewertet und bewertet. Dieser Kriterienkatalog ist in 4 Kategorien eingeteilt. Darunter fallen Kriterien bezüglich der Dokumentation des Frameworks, die mit ihnen verbundenen Entwicklungsprozesse und inwieweit die REST Prinzipien mit Hilfe des Frameworks realisiert werden können. Weiterhin werden erweiterte technische Fähigkeiten betrachtet, wie beispielsweise Entwicklungsunterstützung für transaktionales Verhalten und Asynchronität, aber auch andere wichtige Aspekte für RESTful Applikationen, wie Sicherheit und Zuverlässigkeit.

Inhaltsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Einleitung

1.1. Einleitung

Im Jahr 2000 veröffentlichte Roy Fielding seine Dissertation mit dem Titel "Architectural Styles and the Design of Network-based Software Architectures". In dieser leitet er von verschiedenen Netzwerk-basierten Architekturstilen den "Representational State Transfer" (REST) Stil ab. Dieser Stil, kurz: REST, wird in der Dissertation anhand verschiedener Bedingunen definiert.

Die erste Bedingung ist die Einhaltung einer Client-Server-Architektur. Dabei wird die Benutzerschnittstelle von der Datenhaltung getrennt, um somit Portabilität von der Schnittstelle und die Skalierbarkeit von Servern zu verbessern [Fie00, Abschnitt 5.1.2].
Als zweite Bedingung wird für Client-Server Interaktionen Zustandslosigkeit gefordert. Dadurch sollen Eigenschaften, wie Sichtbarkeit, Zuverlässigkeit und Skalierbarkeit verbessert werden. Die Sichtbarkeit wird unterstützt, indem eine Anfrage zum Server alle nötigen Daten enthalten muss, damit dieser die Anfrage verarbeiten kann. Dadurch können einzelne Anfragen für sich betrachtet werden, ohne dass vorherige Anfragen eine Rolle spielen. Die Zuverlässigkeit wird dann dadurch erhöht, dass beim Ausfall andere Instanzen Anfragen verarbeiten können (Bsp: Hot Pool). Der Skalierbarkeits-Aspekt wird dadurch verbessert, dass der Server keine Zustandsdaten mehr verwalten muss [Fie00, Abschnitt 5.1.3].
Als dritte Bedingung definiert Fielding die Fähigkeit zum Cachen von Antworten (des Servers). Der Vorteil davon ist, dass die Menge potenzieller Anfragen vermindert werden kann. Dabei werden Kriterien, wie Effizienz, Skalierbarkeit und Benutzer empfundene Performanz verbessert, da Anfragen evtl. nicht verarbeitet werden müssen [Fie00, Abschnitt 5.1.4].
Die Forderung nach einer einheitlichen Schnittstelle im REST Architekturstil vereinfacht die Gesamtarchitektur und fördert die Transparenz von Interaktionen [Fie00, Abschnitt 5.1.5].
Eine weitere Forderung besteht daraus, die zu entwerfende Anwendung in verschiedenen Schichten aufzubauen und somit Kapselung von Funktionaliät zu ermöglichen [Fie00, Abschnitt 5.1.6].
Als letzter Aspekt wird Code-On-Demand erwähnt, dieser soll Clients ermöglichen ihre Funktionalität zu erweitern, indem sie Code vom Server laden und ausführen können [Fie00, Abschnitt 5.1.7].

Diese Fachstudie beschäftigt sich mit Frameworks, die das Entwickeln von RESTful Services ermöglichen und erleichtern wollen. Es gibt zahlreiche REST Frameworks auf dem Markt, mit verschiedener Verbreitung, Funktionalität, Lizenzen und in verschiedenen Programmiersprachen [imp].

Diese Fachstudie evaluiert gezielt Open-Source Java Frameworks, die als aktives Projekt bezeichnet werden können. Aktive Projekte sind dabei folgendermaßen zu verstehen: Das Framework wird aktiv weiterentwickelt. Es besteht die Möglichkeit Kontakt zu den Entwicklern/ zur Community des Frameworks aufzunehmen. Das Framework wird von der Communitiy in richtigen Projekten eingesetzt.

Am 13 Februar 2007 wurde dem Java Specification Request Nummer 311 "JAX-RS: The Java API for RESTful Web Services" zugestimmt. Dabei handelt es sich um ein API, dass Unterstützung zur Entwicklung von RESTful Anwendungen im Java Umfeld, bereitstellen soll. Der erste Final Release der JAX-RS 1.0 Spezifikation erfolgte am 10. Oktober 2008, ein Jahr später die Version 1.1 am 23. November 2009. Die ersten, der bis heute aktiven Java-Frameworks entstanden im Jahre 2007, also noch vor der Freigabe von JAX-RS 1.0. Diese waren Apache CXF (Juli 2007), welches aus XFire und Celtix hervorgegangen ist, sowie Restlet (April 2007). In der Folge der Spezifikation wurden viele Frameworks veröffentlicht. Jersey als Referenzimplementierung im Oktober 2008, RESTEasy im Januar 2009, Play! und VRaptor(3.0) jeweils im Oktober 2009 und schließlich Apache Wink im November 2009. 2010 folgten dann noch Resthub (November) und Scooter (Dezember). Die Version 1.1 der JAX-RS Spezifikation wurde am 24. Mai 2013 von der Version 2.0 abgelöst, zu der allerdings noch keine zertifizierten Implementierungen zum Zeitpunkt dieser Fachstudie vorliegen.

1.2. Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 1 – Einleitung: Kapitel 1 stellt das Thema dieser Fachstudie vor.

Kapitel 2 – Verwandte Arbeiten: Kapitel 2 stellt verwandte Arbeiten dieser Fachstudie vor.

Kapitel 3 – Kriterienkatalog: Kapitel 3 stellt den Kriterienkatalog vor, anhand dem die Frameworks ausgewertet und das Bewertungssystem, anhand dessen die Frameworks beurteilt werden.

Kapitel 4 – Bewertungen: In Kapitel 4 werden die einzelnen Frameworks zunächst kurz beschrieben, anhand des Kriterienkatalogs ausgewertet und schließlich beurteilt.

Kapitel 5 – Ergebnisse: In Kapitel 5 werden die Ergebnisse aus Kapitel 4 zusammengefasst, verglichen und ein Fazit gezogen.

Kapitel 6 – Zusammenfassung und Ausblick: fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor.

2. Verwandte Arbeiten

2.1. Guidelines for Designing REST Frameworks

Zuzak beschreibt in dem Artikel den Entwicklungsprozess eines RESTful Systems und versucht anhand von Richtlinien diverse Frameworks zu analysieren. Der Entwicklungsprozess eines Systems wird von Zuzak in drei wichtige Phasen unterteilt. Framework-, Architektur- und Applikationsphase.

Die Frameworkphase beinhaltet die Entwicklung eines Framework-Kerns als Architekturunabhängige Module, die später erforderlich sind für die Implementierung spezifischer Architekturen und Anwendungen. Diese Module beinhalten zum einen generische RESTFul-System-Engines und Repositories zum Verwalten von Protokoll- und Medientyp-Implementierungen.

Architekturphase beinhaltet die Entwicklung der Technologien einer spezifischen RESTful Archtitektur. Hier sollen laut Zuzak, die von der Applikationsphase unabhängigen Protokolle, Medientypen, Verlinkungen und Code-on-Demand Realisierungen implementiert werden.

In der Applikationsphase rät Zuzak, die Client und Server Logik zu implementieren. Dabei kommen Module zum Einsatz, die in der Architektur- und der Frameworkphase entwickelt wurden. In dieser Phase beschäftigen sich die Entwickler des Servers mit der Entwicklung von Ressourcen und deren Identifizierung, während die Client-Entwickler die Applikationslogik entweder als Browser oder als eigene Applikation implementieren. Ebenfalls sollen in dieser Phase Prozessoren für verschiedene Medientypen entwickelt werden, die nicht in während der Architekturphase nicht definiert wurden.

Laut Zuzak sollen Frameworks Modifikationen unterstützen, so dass Entwickler das Framework bei Bedarf anpassen können. Als Beispiel erwähnt Zuzak hier die Definition neuer Protokoll-Header, neue Ressourcen und neue Medientypdefinitionen. Wohlgeformte Schnittstellen und Repository orientiertes Design sind hier von Vorteil. Das Framework soll nach Zuzak mehrere Protokolle der Applikationsschicht verwalten und deren paralleles Benutzen ermöglichen. Hier sollen kleine Module behilflich sein, die verschiedene Aufgaben übernehmen können, wie bspw. die Definition neuer Header oder Verben. Die von Zuzak analysierten Frameworks weisen keinen generischen Ansatz aus, um das Protokoll wechseln zu können. Sie sind fest an HTTP gebunden und bieten keine oder nur sehr kleine Erweiterungsmöglichkeiten. Als nächster Punkt wird URI Design diskutiert. In diesem Bereich soll ein Framework die Möglichkeit anbieten eigene Ressourcen-Identifikatoren, sowie deren Templates zu definieren. Auch die Medientypen sollen erweiterbar, sowie Standards bereits

im Framework enthalten sollen. Als letzter Punkt soll Content-Negotiation vom Framework automatisch durchgeführt werden.

In den weiteren beiden Kapiteln werden Client- und Server-orientierte Richtlinien behandelt. In diesen Kapiteln beschreibt Zuzak das Vorgehen der Entwicklung solcher Systeme. Da diese Fachstudie erster Linie an Frameworks, deren Aufbau und Nutzbarkeit behandelt ist lediglich der erste Teil von [ZS12] relevant. So fanden einige Punkte, Protokollunterstützung und Identifizierungsarten, Einzug in unseren Kriterienkatalog.

2.2. Zuzak NEA

In der Arbeit [ZBD11] beschreibt Zuzak das Vorgehen des Modellierens und des Analysierens von RESTful Software-Systemen. Diese kurze Erläuterung behandelt jedoch lediglich den Teil der Arbeit über das Modellieren von RESTful Software-Systemen mit Hilfe eines nicht deterministischen endlichen Automaten (NEA). Dieser beschreibt, inwiefern sich Aufbau, Zustände und Zustandsübergange eines NEA auf RESTful Applikation übertragen lassen. Zuzak definiert dazu Komponenten oder Module, die bestimmte Operationen zu einer bestimmten Zeit ausführen. So werden Module, wie bspw. Medientyp-Prozessoren definiert, die beim Aufruf einer Ressource weitere Komponenten ansprechen. Speichern von Links, Logik der Applikations-Schicht, Logik der Hypermedia-Schucht und der Protokollschicht werden im nächsten Schritt ausgeführt. Dabei ist entscheidend, in welchem Zustand sich eine Applikation befindet. Bestimmte Zustände markieren dabei Übergänge und somit auch die Nutzung der, eben genannten, drei Komponenten. Zu diesem Zeitpunkt geht es nun in Richtung REST Server zum Request Vorprozessor und/oder zum Request Prozessor, wobei auch an dieser Stelle der Zustand entscheidend ist. Nach diesen Modulen werden noch die letzten Module angesprochen, wie bspw. Code-on-Demand-Engine und das State-Integrator-Modul. Letzteres prüft bspw. ob JavaScript ausgeführt werden soll und übergibt diese dann an die Code-on-Demand-Engine, welche nach der Ausführung des Scriptes den Zustand des NEA wechseln kann. Die Übergänge finden durch Aufrufen, Nachladen und anzeigen von Ressourcen statt. So wird in dieser Arbeit auch der Bezug zur Aussage “Hypermedia as the engine of application state” hergestellt. Die relevanten Kriterien für den Kriterienkatalog wurden hier für den Teil Hypermedia extrahiert. Am Ende von [ZBD11] werden noch Richtlinien für Frameworks beschrieben, die zuvor in [ZS12] erwähnt wurden.

2.3. REST: Die Architektur des Web

In dem Artikel “Die Architektur des Web” versucht Tilkov anhand eines Anwendungsbeispiels das REST Prinzip zu erläutern. Um dieses Beispiel zu entwerfen, bezieht Tilkov sich auf HTTP als Applikationsprotokoll. Verschiedene Aspekte von REST werden erklärt, wie z.B. zustandslose Kommunikation zwischen Client und Server. Weiterhin behandelt Tilkov Ressourcen und deren Identifizierbarkeit. Die einheitlichen Schnittstellen (Uniform Interfaces) müssen demnach bei einer Applikation gut durchdacht sein und sollen von allen Ressource

unterstützt werden. Die HTTP-Verben GET, PUT, POST und DELETE werden von Tilkov in dem Anwendungsbeispiel als Befehle und als solche einheitlichen Schnittstellen bezeichnet. Ein weiterer Aspekt, auf den dieser Artikel eingeht, ist das Hypermedia Prinzip. Wie die Verlinkung, ist auch die Ressource und ihre Identifikation über Ressourcen-Identifikatoren, wie URI beim REST Prinzip im Vordergrund. Als letzten Punkt benennt Tilkov MIME-Types bzw. Medientypen.

2.4. RESTful Web Services Development Checklist

Der Artikel [Vino8] von Vinovski beschreibt die Enwicklung eines RESTful Web Service. Der Artikel aufgebaut wie eine Checkliste. Der Artikel beschreibt grob die wichtigsten Aspekte von REST. Ressourcen und deren Identifikatoren werden erwähnt, sowie die Modellierung der Identifikatoren. Dabei wird in diesem Artikel auf HATEOAS eingegangen. Desweiteren beschäftigt sich Vinovski in diesem Artikel mit Repräsentationen und Medientypen. Er verweist auf die IANA [ian], die für die Verwaltung von globalen Medientypen zuständig sind. Content-Negotiation wird kurz erläutert jedoch nicht weiterführend behandelt. In einem weiterem Kapitel werden HTTP-Methoden präsentiert. Dabei muss beachtet werden, dass eines der vier, für REST wichtigen, Methoden nicht idempotent ist. GET, PUT und DELETE sind idempotent. Ein mehrmaliges Ausführen der Methoden verändert die angesprochene Ressource nur beim ersten mal. Die Antwort auf diese Anfragen ist immer die selbe. Die Methode POST ist nicht idempotent, so dass Vinovski hier auf eine vorsichtige Nutzung aufmerksam macht. Auf weitere Themen geht dieser Artikel leider nicht ein, dennoch sind die gewonnenen Erkenntnisse relevant für den Kriterienkatalog, bezüglich der Unterstützung für Medientypen.

3. Kriterienkatalog

In diesem Kapitel wird der Kriterienkatalog vorgestellt, mit dessen Hilfe die einzelnen Frameworks evaluiert werden. Der Kriterienkatalog ist in vier Kategorien unterteilt: Grundlagen, Entwicklung von REST basierten Anwendungen, Unterstützung grundlegender REST Prinzipien und Erweiterte Technische Fähigkeiten. In dem Kapitel Grundlagen befinden sich Kriterien, wie z.B. die Lizenz, das Vorhandensein von Dokumentation und ob das Framework bspw. die Entwicklung von REST Clients allgemein unterstützt. Die Kriterien unter Entwicklung von REST basierten Anwendungen beschäftigen sich mit Entwicklungsprozessen, die mit dem Framework möglich sind und ob es Möglichkeiten der Modellierung gibt. Unter Unterstützung grundlegender REST Prinzipien werden Kriterien aufgelistet, die sich mit den REST Constraints auseinandersetzen wie bspw. Caching. Bei Kriterien der Kategorie Erweiterte Technische Fähigkeiten werden Fragen bzgl. des HTTP Protokolls gestellt und ob weitere Aspekte, wie Sicherheit und Zuverlässigkeit, aktiv unterstützt werden.

3.1. Grundlagen

Die Kategorie Grundlagen beschäftigt sich mit allgemeinen Eigenschaften des Frameworks und ist in vier Unterkategorien eingeteilt: Allgemeines, REST Server Applikationen, REST Client Applikationen, Architektur und Funktionsweise. Allgemeines besteht aus Kriterien, wie Dokumentation, Lizenz und Community. REST Server Applikationen fragt nach, welche Konfigurationen möglich sind, Server Applikationen zu realisieren. REST Client Applikationen hat Kriterien, die die Fähigkeit zur REST Client Entwicklung evaluieren. Unter der Kategorie Architektur und Funktionsweise, wird die High-level Architektur und die allgemeine Funktionsweise des Frameworks abgefragt.

3.1.1. Allgemeines

1. Gibt es Hilfestellungen für die Entwicklung mit dem Framework? (Tutorials, Codebeispiele, Referenzen)
2. Existiert eine aktive Community?
3. Unter welcher Lizenz steht das Framework zur Verfügung?

4. Ist das Framework prinzipiell erweiterbar, d.h. sind explizite Erweiterungsmechanismen vorgesehen? (Spezielle Erweiterungsmechanismen werden in den entsprechenden Abschnitten separat abgefragt, aber die grundlegende Erweiterbarkeit kann auch interessant sein)

3.1.2. REST Server Applikationen

Jede Software benötigt eine bestimmte Umgebung um lauffähig zu sein. Die Anforderungen reichen dabei von Betriebssystemen, über virtuelle Maschinen bis hin zu bestimmten installierten Anwendungen, die auf dem selben oder angebundenen System laufen müssen. Beispielsweise Server oder Datenbankanwendungen.

1. Ist es möglich, eine Standalone REST Anwendung zu realisieren?
2. In welcher Umgebung kann die implementierte REST Anwendung ausgeführt werden?
3. Wie stark wird die Portierbarkeit der REST Anwendungen unterstützt?

3.1.3. REST Client Applikationen

Eine REST Applikation unterliegt immer der Client-Server Architektur. Diese Fachstudie konzentriert sich zwar hauptsächlich auf Möglichkeiten, die Serverseite einer REST Applikation zu realisieren, dennoch ist auch die Clientseite ein wichtiger Aspekt.

1. Unterstützt das Framework die Entwicklung von Client Applikationen?
2. Welche Hilfsmittel werden von dem Framework, zur Unterstützung der Cliententwicklung, angeboten? (Bspw. Highlevel-HTTP-Aufrufe, Komfortfunktionen etc.)
3. Wie sehen diese genau aus?

3.1.4. Architektur und Funktionsweise

1. Wie ist die Architektur und die grundlegende Funktionsweise des Frameworks?
2. Baut es auf bestehenden Technologien oder Frameworks auf?

3.2. Entwicklung von REST basierten Anwendungen

Bei der Kategorie Entwicklung von REST basierten Anwendungen werden Kriterien in die Unterkategorien Entwicklungsprozess/Vorgehensmodell, Modellierung von REST APIs und Modellierungswerkzeuge eingeteilt. Im ersten Abschnitt werden Kriterien definiert, die ermitteln sollen, ob empfohlene Entwicklungsprozesse existieren, gefordert werden, oder ob Dokumentation zu einem typischen Entwicklungsprozess vorhanden ist. Die Kategorie Modellierung von REST APIs überprüft, ob es Möglichkeiten gibt die API der Anwendung zu modellieren, dabei wird untersucht, inwiefern Interface Description Languages (IDLs) unterstützt werden. Die letzte Unterkategorie Modellierungswerkzeuge beschäftigt sich im Gegensatz zur Modellierung von REST APIs Kategorie mit der Unterstützung von Werkzeugen für die Modellierung, also ob das Framework solche Werkzeuge mitliefert.

3.2.1. Entwicklungsprozess/Vorgehensmodell

Für die Entwicklung von Anwendungen gibt es oft strukturierte Vorgehensweisen, die sich teils erheblich unterscheiden. Ein Beispiel hierzu ist der Top-Down oder Bottom-Up Ansatz bei der Entwicklung WSDL basierter Web Services. REST Frameworks können die Anwendungsentwicklung strukturieren, indem sie bestimmte Vorgehensweisen unterstützen oder auch zwingend voraussetzen.

1. Wird von dem Framework ein explizites Vorgehensmodell gefordert?
2. Werden von dem Framework ein oder mehrere spezifische Vorgehensmodelle unterstützt?
3. Wie sieht der typische Entwicklungsprozess mit dem Framework aus?

3.2.2. Modellierung von REST APIs

Ein weit verbreitetes Hilfsmittel der Softwareentwicklung sind Schnittstellenbeschreibungssprachen (IDLs), mit deren Hilfe die Schnittstellen von Softwarekomponenten beschrieben werden können. Schnittstellenbeschreibungssprachen sind i.d.R nicht abhängig von spezifischen Programmiersprachen.

1. In welchem Umfang werden bestehende IDL von dem Framework unterstützt?
2. Definiert das Framework eine eigene IDL?
3. Falls ja, wie sieht diese aus und welche Funktionalität liefert sie für die Entwicklung einer Applikation?
4. Gibt es in dem Framework Möglichkeiten, bestehende Rest-Applikationen durch eine IDL beschreiben zu lassen, also eine Schnittstellenbeschreibung, bspw. in WADL, automatisch nach Abschluss des Entwicklungsprozesses zu generieren?

5. Können aus einer Schnittstellenbeschreibung in einer IDL (bspw. WADL) Proxies und/oder Stubs generiert werden?

3.2.3. Modellierungswerkzeuge

Es gibt viele Modellierungswerkzeuge, die die Arbeit von Softwareentwicklern zum Teil erheblich vereinfachen können. Diese Werkzeuge bieten Unterstützung zur Modellierung der Architektur, von Use-Cases, bis hin zur Modellierung von Code.

1. Bietet das Framework Möglichkeiten eine zu entwickelnde API oder Anwendung mit Hilfe (graphischer) Tools zu modellieren?
2. Gibt es Möglichkeiten die URI-Struktur einer REST-API zu modellieren?
3. Gibt es Möglichkeiten Ressourcen zu modellieren?
4. Falls ja, wie sieht diese Modellierung aus?
5. Lassen sich damit mehrere Ressourcen und ihre Beziehungen untereinander darstellen?
6. Lässt sich damit das Verhalten von Ressourcen und deren Reaktion auf Operationen modellieren?
7. Gibt es Möglichkeiten Ressourcen-Status-Transitionen im Modell abzubilden?
8. Inwiefern werden Modellierungswerkzeuge unterstützt?

3.3. Unterstützung grundlegender REST Prinzipien

Unterstützung grundlegender REST Prinzipien fasst folgende Unterkategorien zusammen: Ressourcenidentifikation und Ressourcenstruktur, Ressourcentypen, Hypermedia, Medientypen, Caching und Code-on-Demand. Die erste Unterkategorie fasst Kriterien bzgl. der Handhabung von URI's zusammen. Ressourcentypen evaluiert Möglichkeiten, erweiterte Konzepte für REST Ressourcen, wie in [Til09] beschrieben, zu implementieren. Die Hypermedia Kategorie ermittelt ob das Framework den Hypermedia Aspekt von REST vereinfachen kann, bspw. durch Generieren von Links in Repräsentationen. Medientypen evaluiert Content-Negotiation bzgl. verschiedener Medientypen und die Möglichkeit eigene Medientypen einzubringen. Caching greift den gleichnamigen REST Constraint auf, dabei wird gefragt, ob vorgefertigte Mechanismen zum Caching existieren, oder ob den Entwicklern nur die Möglichkeit gegeben wird diese von Hand zu implementieren. Der Code-on-Demand Aspekt wird in der entsprechenden Kategorie abgefragt.

3.3.1. Ressourcenidentifikation und Ressourcenstruktur

Das Konzept von Ressourcen und deren Identifikation ist eines der Kernkonzepte des REST-Architekturstils. Die eindeutige Identifikation von Ressourcen wird dabei i.d.R mit Hilfe von URI (Uniform Resource Identifier) realisiert. [ZS12]

1. Wie erfolgt die Identifikation einzelner Ressourcen auf Entwicklungsebene?
2. Gibt es Unterstützung für URI-Templates?
3. Wie wird das Routing innerhalb des Frameworks umgesetzt?
4. Gibt es im Framework andere Identifizierungsarten außer URI?

3.3.2. Ressourcentypen

Unter Ressourcentypen sind generische Ressourcen zusammengefasst. (Bspw. in abstrakten Klassen). Bei der Entwicklung von REST-Applikationen kann es zu einem großen Implementierungs-Overhead kommen, wenn bspw. immer wieder Listenressourcen "von Hand"geschrieben werden müssen. [ZS12] [Til09]

1. Gibt es vordefinierte Ressourcentypen?

   Beim Entwerfen von Anwendungen gibt es, bedingt durch die fachliche Domäne, immer Aspekte die sich als eigenständige Ressource eignen. Meistens überstimmen Kandidaten mit den fachlichen Kernkonzepten. Diese sind bspw. User in einem Sozialen Netzwerk, Threads in einem Forum. Diese Kandidaten sollten eine wichtige Rolle beim Entwurf spielen und somit als Primärressourcen gelten.

2. Unterstützt das Framework beim Entwurf von Primärressourcen?

   Unter einer Subressource versteht sich eine Ressource die Teil einer anderen ist. Beispiele hierfür sind eine Bestellung in einer Bestellliste, die Adressen einer Bestellung oder Lieferanten einer Bestellung

3. Unterstützt das Framework beim Entwurf von Subressourcen?

   Eine Listenressource ist eine Ressource die eine Menge von Ressourcen des selben Typs zusammenfasst.

4. Unterstützt das Framework beim Entwurf von Listenressourcen ?

   Projektionen sind Ressourcen die dazu dienen Informationsmengen von anderen Ressourcen einzuschränken, um übertragene Datenmengen zu reduzieren.

5. Unterstützt das Framework beim Entwurf von Projektionen ?

   Aggregationen fassen unterschiedliche Attribute der Primär- oder Listenressourcen zusammen. So ist es möglich die Anzahl der Interaktionen zwischen dem Client und dem Server zu begrenzen.

6. Unterstützt das Framework das Entwerfen solcher Aggregations-Ressourcen?

   Einzelne Schritte oder ganze Arbeitsaufträge können als eine Ressource zusammengefasst werden. Solche Ressourcen werden auch Aktivitäten genannt.

7. Werden Aktivitäten von dem Framework unterstützt?

3.3.3. Hypermedia

Hypermedia ist eines der Kern-Konzepte der REST Architektur (HATEOAS), hierbei handelt es sich darum, dass der Server dem Client anhand von Meta-Daten vermittelt, welche Äktionenïnnerhalb seines Zustandes möglich sind. Frameworks sollten daher Mechanismen anbieten die es für Entwickler erleichtern, diese Aktionen auf Ressourcen, abhängig vom Zustand abzubilden. [ZS12] [Til09] [ZBD11] [kar]

1. Gibt es Möglichkeiten zustandsabhängig Links zu Repräsentationen hinzuzufügen?
2. Gibt es Möglichkeiten Link-Relationen auch für Medientypen zu definieren die nicht aus dem Hypertext-Umfeld kommen ?
3. Gibt es Komfort-Funktionen die das Verknüpfen von Inhalten erleichtern?
4. Gibt es vordefinierte "Verlinkungs-Typen". Bspw. Verlinkung auf Editoren. (siehe googleDocs)
5. Gibt es die Möglichkeit in vorgegebenem Rahmen Verlinkungs-Typen selbst zu definieren?
6. Werden Link-Standards unterstützt ?
7. Unterstützt das Framework Konzepte und Techniken des Semantic Web, also bspw. Ontologien oder RDF?

3.3.4. Medientypen

Verschiedene Medientypen sind wichtig für ein REST Framework. So muss ein Framework mehrere Medientypen beherrschen. Beispiele dafür sind HTML, ATOM, XML, CSV etc..

1. Welche Medientypen werden von dem Framework aktiv unterstützt?
2. Besteht eine Möglichkeit eigene Medientypen zu definieren? Wie wird dies realisiert?
3. Wird Client-side Content-Negotiation unterstützt?
4. Wird Server-side Content-Negotiation unterstützt?
5. Wie werden Medientypen bzgl. Content-Negotiation gewählt?
6. Wie werden verschiedene Codierungen bzgl. der Content-Negotiation gewählt?
7. Wie werden verschiedene Zeichensätze bzgl. der Content-Negotiation gewählt?

8. Wie werden verschiedene Sprachen bzgl. der Content-Negotiation gewählt?

3.3.5. Caching

Eines der Schlüsselziele von Rest ist auch hohe Skalierbarkeit. Elementar für die Skalierbarkeit ist dabei das Cachen.

1. Inwieweit wird Caching von einzelnen Frameworks unterstützt?
2. Gibt es voreingestellte Settings, die man nutzen kann? Oder muss man bei jeder erzeugten Nachricht Caching-Header "von Hand" setzen?
3. Welche Caching-Modelle werden unterstützt?
4. Werden E-Tags unterstützt?
5. Automatische Generierung von E-Tags?

3.3.6. Code-On-Demand

Mit dem Code-On-Demand-Aspekt beschreibt Fielding in seiner Dissertation die Möglichkeit, die Funktionalität des Clients mittels runterladbarem und ausführbaren Code zu erweitern.

1. Inwiefern bietet das Framework Unterstützung zur Realisierung des Code-On-Demand Aspekts?

3.4. Erweiterte Technische Fähigkeiten

Unter Erweiterte Technische Fähigkeiten wird die Unterstützung zur Realisierung wichtiger Eigenschaften einer Applikation, wie Sicherheit und Zuverlässigkeit, untersucht. Weiterhin wird untersucht, in welchem Umfang das HTTP Protokoll genutzt werden kann. Es wird anhand folgender Unterkategorien eingeteilt: Protokollunterstützung jenseits von HTTP, HTTP, Unterstützung für Transaktionen, Security, Asynchronität, Zuverlässigkeit und Umgang mit großen Daten. Protokollunterstützung jenseits von HTTP fokussiert dabei auf die Möglichkeit, mit dem Framework auch andere Netzwerk-Protokolle außer HTTP zu nutzen, wobei sich die Kategorie HTTP mit dem gleichnamigen Protokoll auseinander setzt. Unterstützung für Transaktionen beschäftigt sich mit der Fähigkeit, transaktionales Verhalten zu realisieren. Asynchronität versucht zu ermitteln, ob Entwickler asynchrone Programmiermodelle verwenden können und ob zwischen Client und Servern asynchrone Kommunikation genutzt werden kann. Zuverlässigkeit geht auf Fähigkeiten ein, Fehlverhalten bei der Client-Server-Kommunikation deterministisch zu behandeln. Bei der Kategorie Umgang mit großen Daten wird untersucht, ob das Framework Hilfestellung, zum Down- und Upload großer Daten, bietet.

3.4.1. Protokollunterstützung jenseits von HTTP

Der REST Architekturstil wurde ursprünglich konzipiert um vorhandene Probleme in HTTP 1.0 zu lokalisieren [Fie00]. Diese wurden dann mit HTTP 1.1 behoben. Trotz der Nähe zwischen REST und HTTP gibt es spezielle Protokolle für REST (CoAP, WAKA), die HTTP ersetzen sollen/ können.

1. Welche Protokolle werden, abgesehen von HTTP, von dem Framework unterstützt?
2. Wie einfach ist es zwischen verschiedenen Protokollen zu wechseln?
3. Ist die gleichzeitige Nutzung verschiedener Protokolle möglich?
4. Können angebotene Protokolle in vollem Umfang genutzt werden?
5. Falls Nein: Ist es möglich die Protokoll-Implementierungen entsprechend zu erweitern?

3.4.2. HTTP

HTTP ist das Referenzprotokoll für REST-Applikationen. In welchem Ausmaß nutzen die Frameworks das häufig genutzte Protokoll? Wenn HTTP genutzt werden soll, müssen auch HTTP-Verben unterstützt werden. [Til09] [FGM+99]

1. Werden alle HTTP-Verben unterstützt?
2. Erlaubt das Framework die Nutzung bestimmter Verben ohne Einhaltung der spezifizierten Verbsemantik wie Idempotenz und Sicherheit?

Mit Verwendung von HTTP müssen auch Header unterstützt werden.

1. Können alle Headerfelder gesetzt und gelesen werden?
2. Gibt es Möglichkeiten vordefinierte Headersets zu benutzen, oder die Möglichkeit solche anzulegen?

3.4.3. Unterstützung für Transaktionen

Eine Transaktion ist eine Sequenz von Verarbeitungsschritten, die als Einheit betrachtet wird und die ACID-Eigenschaften erfüllt. D.h. entweder werden alle Verarbeitungsschritte durchgeführt, oder gar keine. Treten mitten in der Verarbeitung Fehler auf werden alle vorherigen zur Transaktion zugehörigen, bereits ausgeführten Schritte zurückgesetzt (Rollback). In dieser Kategorie geht es darum, ob das Framework aktiv Unterstützung anbietet, indem bspw. eine Transaction-API angeboten wird.

1. Wird transaktionales Verhalten aktiv von dem Framework unterstützt?
2. In Form von transaktionalen Aufrufen? Also in der Zeit zwischen dem Erhalten eines Requests und dem Senden der Antwort?

3. Unterstützt das Framework die Integration bestehender Transaktionstechniken wie bspw. JTA?
4. Gibt es Unterstützung für 2PC (Two-phase commit protocol)?

3.4.4. Security

In verteilten Systemen und Anwendungen spielt das Thema Sicherheit eine große Rolle. Beim Einsatz von REST Applikationen kommuniziert i.d.R. eine Client-Anwendung mit einem dazugehörigen Server. Um zu verhindern, dass bei der Kommunikation über das Internet ein unautorisierter Zugriff auf die gesendeten Nachrichten oder gespeicherten Ressourcen stattfindet, bedarf es Sicherheitsmechanismen wie SSL und Authentifizierung.

1. Kann das Framework SSL-Unterstützung anbieten? Kann ggf. ein Benutzer-Zertifikat überprüft werden?
2. Werden Authentifizierungen (Im HTTP Umfeld bspw. Basic-Auth) unterstützt? Wie? Welche?
3. Kann eine, mit Hilfe des Frameworks entwickelte, Standalone Anwendung uneingeschränkt konfiguriert werden?
4. Gibt es Möglichkeiten Nachrichten automatisch zu verschlüsseln?
5. Gibt es Möglichkeiten Nachrichten automatisch mit einer Signatur zu versehen?

3.4.5. Asynchronität

Asynchrone Kommunikation ermöglicht Servern, Anfragen nach eigenem Ermessen zu bearbeiten. Clients profitieren von der Asynchronität indem sie nicht blockiert werden, da sie nicht auf Antworten warten müssen.

1. Werden asynchrone Programmiermodelle von dem Framework aktiv unterstützt? (Bspw.Threads, Non-blocking I/O)
2. Gibt es Möglichkeiten asynchronen Nachrichtenaustausch zu ermöglichen? (Bspw. Patterns wie Polling etc.)

3.4.6. Zuverlässigkeit

Um zuverlässig zu arbeiten, muss ein Client stets wissen, ob seine Anfragen richtig bearbeitet wurden oder nicht. Bleiben einzelne Anfragen ohne Antwort, muss der Client entscheiden, ob er die Anfrage erneut schickt oder nicht. Idempotente Anfragen können dabei jederzeit neu gesendet werden. Um Zuverlässigkeit in REST-Applikationen zu garantieren, müssen kritische Fehlerfälle betrachtet werden. Dies ist der Fall, wenn ein Client vom Server keine Antwort erhält:

Die Anfrage wurde vollständig bearbeitet, die Antwort kommt aber nicht beim Client an.
Die Anfrage wurde nicht vollständig bearbeitet, es kommt aber keine Antwort(auch keine Fehlermeldung) beim Client an.
Das Problem ist, dass der Client nun entscheiden muss, ob er die Anfrage neu sendet oder nicht. Dies ist vor allem kritisch bei nicht idempotenten Methoden.

1. Inwiefern unterstützt das Framework den Entwickler um oben beschriebenes Problem zu lösen?
2. Gibt es PUT/POST Kombinationen? [Tilo9]
3. Reliable POST Unterstützung? [Tilo9]

3.4.7. Umgang mit großen Daten

In der heutigen Online Welt werden viele Daten bzw. Dateien zwischen Client und Server ausgetauscht. Es ist auch möglich, dass es nicht nur bei kleinen JSON Objekten oder ähnlichem bleibt.

1. Unterstützt das Framework den Austausch großer Dateien?
2. Wird ein Vorgehen vorgegeben, um einen solchen Mechanismus selbst zu implementieren, falls nicht vorhanden?

3.5. Bewertungssystem

Die Bewertung der untersuchten Frameworks erfolgt einzeln für jede der Kategorien des Kriterienkatalogs. Die Kategorie Architektur und Funktionsweise wird dabei nicht berücksichtigt.

Auf die Errechnung von Gesamtnoten für Kapitel oder Frameworks wird verzichtet. Der Grund dafür liegt darin, dass diese Fachstudie nicht darauf ausgerichtet ist ein Ranking zu erstellen, sondern Hilfestellung zur Auswahl eines Frameworks leisten soll, das den Zielen eines Softwareprojekts dient. Desweiteren ist es nicht möglich die richtige Gewichtung einzelner Bewertungen festzulegen, zum einen, weil sich die untersuchten Kriterien sehr stark unterscheiden, zum anderen, weil die individuellen Anforderungen an ein Framework von Entwickler zu Entwickler und von Projekt zu Projekt unterschiedlich sind.

Zur Anwendung kommt ein 3-stufiges Bewertungssystem beginnend bei bei der Stufe 0 bis zur Stufe 2.

Stufe 2 Stufe 2 bedeutet, dass in dem bewerteten Bereich die meisten Anforderungen erfüllt sind und dass die Entwicklung in diesem Bereich die Entwicklung sehr komfortabel möglich ist.

Stufe 1 Stufe 1 bedeutet, dass in dem bewerteten Bereich die meisten Anforderungen erfüllt sind, sich die Entwicklung allerdings schwieriger gestaltet, weil keine Highlevel-Zugriffe oder Komfortfunktionen vorhanden sind.

Stufe 0 Die Stufe 0 kann 2 Dinge bedeuteten. Entweder wurden die Anforderungen in dem bewerteten Bereich nicht erfüllt, oder es war keine ausreichende Dokumentation in diesem Bereich vorhanden um eine Aussage zu treffen.

4. Bewertungen

4.1. Jersey

Eckdaten

URL	http://jersey.java.net/
Lizenz	CDDL 1.1, GPL v2
Entwickler	Oracle

Tabelle 4.1.: Eckdaten Jersey

Beschreibung Jersey gilt als Referenz-Implementierung für JAX-RS (JSR 311) [jaxa]. Jersey implementiert die in JAX-RS definierten Annotationen, die es erlauben URIs auf Klassen und Methoden zu mappen, denen wiederrum Annotationen zur verfügung stehen um festzulegen auf welches HTTP Verb diese reagieren.

Beispiel In JAX-RS wird alles mittels Annotationen deklariert. Das Mapping von URIs auf Methoden/Klassen geschieht mittels @Path. Um nun an einem Pfad HTTP Methoden zu erlauben werden Annotation @GET, @PUT, etc. entsprechend den HTTP Verben deklariert. Medientypen werden mittels @Produces/@Consumes-Annotation festgelegt, wobei Produces für den Medientyp in der Antwort und Consumes für Medientypen bei der Anfrage steht. Ein Beispiel ist in Listing 4.1.

4.1.1. Grundlagen

Abschnitt	**Stufe**	**Kommentar**
Allgemeines	2	Ausführliche Dokumentation, aktive Community, Referenzimplementierung von JAX-RS, gut erweiterbar
REST Server Applikationen	1	Auf Servlet ausgelegt, Standalone möglich
REST Client Applikationen	2	Client-API, Komfortfunktionen, gute Erweiterungsmöglichkeiten

Tabelle 4.2.: Auswertung Jersey Grundlagen

Listing 4.1 JAX-RS Code Beispiel

```
package com.sun.ws.rest.samples.helloworld.resources;
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;

// The Java class will be hosted at the URI path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

    // The Java method will process HTTP GET requests
    @GET
    // The Java method will produce content identified by the MIME Media
    // type "text/plain"
    @Produces("text/plain")
    public String getClichedMessage() {
        // Return some cliched textual content
        return "Hello World";
    }
}
```

Allgemeines

1 Auf der offiziellen Website existiert ein User Guide, welches 13 Themen bearbeitet. Hier wird auch ein schneller Einstieg geboten in dem Kapitel „Getting Started“. Nach einigen Schritten ist eine HelloWorld Applikation bereits aufgesetzt und kann getestet werden. Der User Guide bietet zu vielen behandelten Themen auch einige Code Beispiele an, so dass ein Einstieg möglich ist. Außer dem User Guide existiert noch eine Wiki Platform für Jersey. Hier gibt es ebenfalls einen Getting Started Kapitel wie in dem offiziellem User Guide. Ebenfalls zu finden sind hier unterschiedliche Beispiele.

21 Projekte sind auf der Jersey Website aufgeführt, welche Jersey nutzen.

Die Hilfestellung ist für das Framework Jersey auf jedenfall gegeben, wie relevant und ausführlich die Tutorials und Codebeispiele sind, kann in dieser Arbeit jedoch nicht herausgearbeitet werden.

2 Neben den offiziellen Mailinglisten vom Entwickler Oracle, wird über Jersey auf vielen sozialen Plattformen diskutiert. So bietet Jersey mit Git-Hub einen Zugang zu dem Quellcode. Auf Git-Hub wird im Falle von Fragen auf die Website stackoverflow.com verwiesen, Jersey hat hier einen eigenen Tag „jersey“.

3 Jersey steht unter den Lizenzen COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0 und der GNU General Public License (GPL) Version 2, June 1991 zur Verfügung .

4 Da Jersey die Referenzimplementierung von JAX-RS 1.1 ist, gibt es die Erweiterungsmechanismen aus JAX-RS. Die API ermöglicht es neue MessageBodyReader und MessageBodyWriter zu integrieren, wobei erstere dafür zuständig sind HTTP Requestbodies zu deserialisieren und letztere zum serialisieren. ExceptionManager werden dazu verwendet Java-Exceptions auf HTTP Responses abzubilden. Zusätzlich existiert die Möglichkeit ContextResolver zu nutzen um bspw. einen JAXBContext für bestimmte jaxb-annotierte Java Typen zu konfigurieren und diese beim De-/Serialisieren zu verwenden.

REST Server Applikationen

1 Es ist möglich eine Standalone REST Anwendung mit den reinen Jersey Bibiliotheken zu realisieren. Dabei kann man den im JDK mitgelieferten HTTP Server verwenden oder beispielsweise den Grizzly HTTP Server. Jersey bietet für diese und andere verschiedene Module an, um sie nutzen zu können.

2 Sofern die Anwendung als Standalone implementiert ist benötigt diese keine Umgebung. Ansonsten kann diese Anwendung in einem Servlet Container ausgeführt werden, der mindestens Servlet Spezifikation 2.5 erfüllt.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

REST Client Applikationen

1 Ja, das Jersey Framework enthält eine Client-API. Die Client-API nutzt HttpURLConnection oder den Apache HTTP client. Diese beiden werden von Jerseys Client-API gewrapped. Es gibt Unterstützung für automatische (De-)Serialisierung (mittels JAXB). Weiterhin werden auch Filter unterstützt.[jera]

2 Die Client-API von Jersey bietet Highlevel-HTTP-Aufrufe und automatische (De-)Serialisierung von Objekten.

3 Die HTTP-Aufrufe können mit einfachen HTTP-Methoden auf einem WebResource Ojekt ausgeführt werden. Für wichtige Header-Felder(wie bspw. Accept-Header) gibt es ebenfalls Methoden. Die automatische (De-)Serialisierung von Objekten wird wie beim Server auch von MessageBodyReader/-Writer übernommen. Weiterhin besteht auch die möglichkeit ClientFilter zu implementieren, die in der Lage sind Antworten zwischen Client und Server manipulieren. Beispielsweise können hier auch weitere Header gesetzt werden(z.B zur Authentifizierung).

Architektur und Funktionsweise

1 Jersey nutzt Java-Annotationen um HTTP-Verben, Medientypen auf Methoden zu mappen. Dabei werden bspw. Klassen mit dem @Path Attribut bei der Klassendeklarationen als Rootklassen inkl. entsprechendem Pfad. mittels den Annotationen @GET, @PUT etc. werden dann einzelne HTTP-Anfragen auf die entsprechenden methoden gemapped.

2 Für das verwenden von Jersey in einem Container wird Servlet 2.5 vorrausgesetzt, kann aber auch Standalone mit dem in Java mitgelieferten HTTP Server verwendet werden.

4.1.2. Entwicklung von REST basierten Anwendungen

Abschnitt	Stufe	Kommentar
Entwicklungsprozess/ Vorgehensmodell	0	
Modellierung von REST APIs	1	Laufzeitgenerierung von WADL zu einzelnen Ressourcen
Modellierungswerkzeuge	0	

Tabelle 4.3.: Auswertung Jersey Entwicklung von REST basierten Anwendungen

Entwicklungsprozess/Vorgehensmodell

1 Nein.

2 In Jersey existieren keine speziellen Mechanismen, die die Entwicklung mittels einem bestimmten Vorgehensmodell explizit bevorteilen.

3 Der typische Entwicklungsprozess mit Jersey kommt dem Wasserfall-Modell am nächsten. Die wichtigste Aufgabe, um spätere Korrekturarbeiten zu vermeiden ist es zunächst, alle Ressourcen und der Anordnung in der URI-Struktur zu identifizieren und ein geeignetes Java-Modell zu implementieren. Mit diesen Voraussetzungen können dann die Ressourcenklassen und Geschäftslogik implementiert werden.

Modellierung von REST APIs

1 Jersey unterstützt die automatische Generierung von WADL.

2 Nein.

3 Jersey kann zur Laufzeit einer Restapplikation eine WADL Beschreibung generieren. Diese kann einfach mittels **HTTP GET http://pfad.zu.deiner.restapplikation/application.wadl** (Bsp.: http://localhost:8080/application.wadl) generiert werden. Für eine einzelne Ressource kann WADL mittels HTTP OPTIONS auf dieser Ressource generiert werden. Zusätzlich bietet Jersey die Möglichkeit doc Elemente oder javadoc der Javaklassen in eine Datei zu schreiben, so dass sie benutzt werden können um die WADL zu erweitern. Ebenfalls gibt es die Möglichkeit mittels Maven-wadl-plugin WADL aus dem SourceCode zu generieren, ohne dass die Applikation aktiv ist. [jerb]

4 Jersey selbst unterstützt das Generieren von Java aus WADL nicht. Allerdings kann mittels dem wadl2java Tool von GlassFish ClientCode generiert werden, der die Jersey-API nutzt. Es wird dabei eine Klasse pro root-resource angelegt. Genutzt wird die Jersey 1.x und JAX-RS 2.0 Spezifikation. [jerc]

Modellierungswerkzeuge

1 Nein.

2 Nein.

3 Nein.

4 Kann nicht beantwortet werden.

5 Nein.

6 Nein.

7 Nein.

8 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.1.3. Unterstützung grundlegender REST Prinzipien

Abschnitt	Stufe	Kommentar
Ressourcenidentifikation und Ressourcenstruktur	1	JAX-RS 1.1 Annotationen
Ressourcentypen	0	
Hypermedia	1	Setzen von Link-Headern mittels Annotation möglich
Medientypen	1	JAX-RS 1.1
Caching	1	CacheControl Klasse als Abstraktion von Caching-Headern
Code-On-Demand	0	

Tabelle 4.4.: Auswertung Jersey Unterstützung grundlegender REST Prinzipie

Ressourcenidentifikation und Ressourcenstruktur

1 Gemäß JAX-RS erfolgt die Identifikation über URI-Elemente, die per @Path Annotation an Klassen und Methoden gebunden werden können. Dabei können sowohl einzelne URI-Elemente, als auch Teilpfade einer URI verwendet werden. Es ist damit möglich Aufruf-Hierarchien über verschiedene Klassen und Methoden hinweg zu realisieren.

2 Ja. Jersey unterstützt Variablen in Pfaden. Auf diese Variablen kann wiederum mittels Annotationen zugegriffen werden. [jerd] [jere]

3 Siehe dazu JAX-RS 1.1 Spezifikation [jaxa, JAX-RS 1.1 Abschnitt 3.7]

4 Nein.

Ressourcentypen

1 Primärressourcen und Subressourcen können mit Mitteln aus JAX-RS 1.1 unkompliziert realisiert werden, sind aber nicht speziell als solche deklariert.

2 Gemäß JAX-RS 1.1 können Klassen mittels @Path Annotationen als Primärressource verfügbar gemacht werden.

3 Gemäß JAX-RS 1.1 können Methoden innerhalb von Klassen mittels @Path Annotationen als Subressourcenidentifikator genutzt werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Hypermedia

1 Ja. mittels der @Ref Annotation lassen sich Links in Repräsentationen einfügen. Dabei injiziert die Jersey-Runtime die passende URI in das JavaObjekt, bevor es durch einen MessageBodyWriter serialisiert wird. Zustandsabhängige Link-Injektion kann über die condition Einstellung realisiert werden. (@Ref ist nicht Teil der JAX-RS 1.1 Spezifikation) [jerf]

2 Ja, Jersey implementiert dabei RFC 5988 [rfc] um Links per HTTP Header mitzuliefern. Dies wird auch über die @Ref Annotation realisiert.

3 Abgesehen von der @Ref-Annotation gibt es keine weiteren. „komfort-Verknüpfungen“

4 Es gibt keine vordefinierten „Verlinkungs-Typen“.

5 Mittels der experimentellen Annotationen @Action, @ContextualActionSet und @HypermediaController lassen sich beispielsweise Aktionen als Subressourcen implementieren. Mittels @ContextualActionSet lassen sich die jeweils verfügbaren Aktionen dynamisch einschränken.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Jersey unterstützt keine Konzepte und Techniken des Semantic Web. Es ist jedoch möglich Jersey gemeinsam mit Som(m)er [sum] zu benutzen, um die Java Objekte nach RDF zu serialisieren.

Medientypen

1 Jersey als Referenzimplementierung von JAX-RS, bietet text/xml, application/xml und application/- *+xml als nutzbare Medientypen, zusätzlich können durch die generischen Provider alle möglichen Medientypen verarbeitet (*/*) werden. Dabei werden die Messagebodies in Strings, InputStream, etc.geladen. [jerg]

2 Die JAX-RS Spezifikation fordert das neue Medientypen registriert werden können. So können Provider implementiert werden die das Serialisieren/Deserialisieren von Medientypen übernehmen. Dafür muss ein Interface implementiert werden (*MessageBodyReader* bzw. *MessageBodyWriter*) und mit einer *@Provider* Annotation versehen werden.

3 Client-side Content-Negotiation ist durch Accept-Header realisiert, die sich mittels Methoden setzen lassen.

4 Server-side Content-Negotiation wird mittels @Produces und @Consumes bewältigt. @Produces-Einträge lassen sich auch priorisieren. [jerh]

5 Siehe [jaxa, JAX-RS 1.1 Abschnitt 4.2.1]

6

- charset=UTF-8 wird standard-mäßig gesetzt.
- Client-side: Response.ok(entity).header("charset", ütf-8").build(). So lassen sich alle möglichen Header setzen.
- Server-side: Durch die @Produces-Annotation laesst sich das charset auch beeinflussen.
- *@Produces*(*MediaType.TEXT_HTML*, *MediaType.ACCEPT_CHARSET* = *"UTF* − 8")

7 Die ResponseBuilder-Methode "language" erwartet ein java.util.Locale [jeri] WebResource.acceptLanguage(java.util.Locale loc). [jerj]

8 Das ist den Entwicklern überlassen, dabei müssen sie die Header des Requests selbst auslesen und entsprechende Repräsentationen liefern.

Caching

1 Es gibt vorgefertige Klassen um Caching zu realisieren. CacheControl ist eine Abstraktion von den Caching-Headern in HTTP. EntityTag existiert als Abstraktion von EntityTags. Mit diesen kann ein Entwickler das gewünschte Cachingverhalten implementieren.

2 Es existieren keine Möglichkeiten CachingHeader ohne direkten Einfluss des Entwicklers zu setzen. Jegliche Logik dies bezüglich muss vom Entwickler selbst implementiert werden.

3 Expirationsmodell und Validierungsmodell können von Entwicklern selbst implementiert werden über die bereitgestellte API.

4 Ja.

5 Nein.

Code-On-Demand

1 Es gibt keine vorgefertigten Möglichkeiten den Code-On-Demand Aspekt gezielt zu nutzen.

4.1.4. Erweiterte Technische Fähigkeiten

Abschnitt	Stufe	Kommentar
Protokollunterstützung jenseits von HTTP	0	
HTTP	2	JAX-RS 1.1
Unterstützung für Transaktionen	0	
Security	1	SSL
Asynchronität	1	Non-Blocking Client
Zuverlässigkeit	0	
Umgang mit großen Daten	2	Extra Annotation

Tabelle 4.5.: Auswertung Jersey Erweiterte Technische Fähigkeiten

Protokollunterstützung jenseits von HTTP

Protokollunterstützung jenseits von HTTP

1 Nein.

2 Kann nicht beantwortet werden.

3 Nein.

4 HTTP kann in vollem Umfang genutzt werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

HTTP

1 JAX-RS 1.1 definiert als Annotationen GET, PUT, POST, DELETE und HEAD. Desweiteren verlangt JAX-RS 1.1 auch noch Antworten auf OPTIONS Anfragen. Weiterhin können über die Annotation HttpMethod weitere HTTP-Methoden gemapped werden.

2 Da die HTTP-Verb-Annotationen lediglich zur Auswahl von Java-Methoden verwendet werden, bleibt die Einhaltung der Verbsemantik in Bezug auf Idempotenz und Sicherheit in der Verantwortung des Entwicklers.

1 Ja.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Unterstützung für Transaktionen

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Kann nicht beantwortet werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Security

1 Jersey unterstütz SSL. Dabei ist es irrelevant ob die Applikation ein standalone Server ist oder in einem Container deployed wird. Benutzerzertifikate können mittels @Context Annotation überprüft werden. Hierbei wird die ganze Anfrage in den jeweiligen Kontext injiziert.

2 Die Konfiguration hängt vom genutzen Server ab (JDK HTTPServer, Grizzly HTTP Server).

3 Nein, bis auf die Möglichkeit SSL zu nutzen, muss alles selbst entwickelt werden.

4 Nein, alles muss selbst entwickelt werden, dabei empfehlen sich eigene MessageBody-Reader und Writer.

5 Nein.

Asynchronität

1 Jersey bietet für Clients den nicht-blockierenden Jersey Client, um dadurch die Vorteile von asychronität zu bekommen. Server-seitig gibt es keine Unterstützung, da JAX-RS 1.1 konzeptionell auf synchronem Request-Response basiert.

2 Es gibt keine vorgefertigten Mechanismen für asynchronen Nachrichtenaustausch, alle müssen vom Entwickler selbst implementiert werden.

Zuverlässigkeit

1 Gar nicht.

2 Nein.

3 Nein.

Listing 4.2 Erstellung einer Applikation in Scooter.

```
1. Erstelle und definiere eine Datenbank
CREATE DATABASE ...;
USE ...;
CREATE TABLE entries (...);

2. Erstelle die Applikation
scooter> java -jar tools/create.jar customerservice

3. Generiere das Geruest
scooter> java -jar tools/generate.jar customerservice scaffold entry


4. Starte deine Applikation
scooter> java -jar tools/server.jar customerservice
```

Umgang mit großen Daten

1 Ja, Jersey besitzt die Annotation „FormDataParam“ die speziell für den Austausch von Dateien gedacht ist. In Verbindung mit dem Medientyp „multipart/form-data“ wird der Austausch von großen Dateien unterstützt.

2 Nein.

4.2. Scooter

Eckdaten

URL	http://www.scooterframework.com/
Lizenz	LGNU Library or Lesser General Public License version 3.0 (LGPLv3)
Entwickler	Amazing Force (http://www.amazingforce.com/)

Tabelle 4.6.: Eckdaten Scooter

Beschreibung Geeignet für Browserbasierte WebApplikationen, „ControllerKlassen“(POJO) mit public-Methoden, Routing durch .properties.

Beispiel Eine Applikation wird bei Scooter schon nach 3 Schritten lauffähig. In 4.2 werden die Schritte aufgeführt, um eine Applikation in wenigen Minuten lauffähig zu machen.

4.2.1. Grundlagen

Abschnitt	Stufe	Kommentar
Allgemeines	1	Gute Dokumentation, semi-aktive Community, erweiterbar
REST Server Applikationen	1	Auf Standalone ausgelegt, Servlet aufwändig
REST Client Applikationen	0	keine dedizierte Client-API, auf Browser-Clients ausgelegt

Tabelle 4.7.: Auswertung Scooter Grundlagen

Allgemeines

1 Ja, es werden drei Beispiele vorgestellt darunter ein Blog und eine Twitter ähnliche Applikation, es gibt Hilfestellungen zum Installieren, zur Projektstruktur und zum Design der eigenen Applikation.

2 Es gibt eine Google Group [scoa] und einen Twitter Account [scob].

3 Das Scooter Framework gibt es unter der LGPL [lgp].

4 Scooter besitzt einen Plugin Mechanismus um beispielsweise Caches, Content-Handler für Medientypen und Templates für Views hinzuzufügen.

REST Server Applikationen

1 Ja, für Scooter wird sogar empfohlen die Anwendungen standalone zu realisieren.

2 Scooter ist darauf ausgelegt Applikationen standalone zu entwickeln, zusätzlich existiert ein Ant-Skript mit dem es möglich ist diese in eine WAR Datei zu packen und somit auf Servern zu verwenden die WAR Dateien verstehen.

3 Gar nicht bzw. schwer evaluierbar.

REST Client Applikationen

1 Ja.

2 Es ist nicht möglich dedizierte Clients in Scooter zu entwickeln.

3 Scooter stützt sich auf das MVC Pattern und bietet speziell für die View vorgefertigte Templates an. Diese werden anhand der Backend konfiguriert, bspw. wird eine HTML-Seite generiert falls eine neue Klasse im Datenmodell angelegt wird. Die Templates für die View sind erweiterbar man kann auch sogenannte TemplateHandler als Plugin einfügen. Zur reinen Client-Entwicklung ist Scooter eher ungeeignet.

Architektur und Funktionsweise

1 Scooter versucht die Prinzipien aus Ruby-on-Rails in Java zu ermöglichen. Dabei wird übergeordnet das MVC-Pattern genutzt. Für das Modell wird ausschliesslich das ActiveRecord Pattern verwendet, dafür wird eine Klasse angeboten von denen Modellklassen erben können. Die Modellklassen werden automatisch mit der angebundenen Datenbank konsistent gehalten, auch Änderungen auf den Klassen werden beachtet. Für die View werden vorgefertigte Templates verwendet die dann Anhand des Datenmodells und der Control HTML Seiten generiert, bspw. werden Forms generiert falls die Control es erlaubt bestimmte Daten zu ändern. Die Control wird mittels Routing Dateien und Controller-Klassen implementiert, dabei kann zwischen verschiedenen Ansätzen gewählt werden. Der RestfulRequestProcessor erlaubt HTTP Methoden auf URI's innerhalb der Applikation zu mappen, dabei werden dann interne Mechanismen/Konventionen von Scooter für die jeweilige Semantik verwendet, bspw. würde ein GET auf ./posts/123 den Post 123 eines Blogs zurückgeben und ein POST auf ./posts/ einen neuen Post erstellen [scoc].

2 Scooter baut auf Java, Servlets und einem internen Jetty auf.

4.2.2. Entwicklung von REST basierten Anwendungen

Abschnitt	Stufe	Kommentar
Entwicklungsprozess/ Vorgehensmodell	0	
Modellierung von REST APIs	0	
Modellierungswerkzeuge	0	

Tabelle 4.8.: Auswertung Scooter Entwicklung von REST basierten Anwendungen

Entwicklungsprozess/Vorgehensmodell

1 Nein.

2 Nein.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Modellierung von REST APIs

1 Gar nicht.

2 Nein.

3 Nein.

4 Nein.

Modellierungswerkzeuge

1 Nein.

2 Nein.

3 Nein.

4 Kann nicht beantwortet werden.

5 Nein.

6 Nein.

7 Nein.

8 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.2.3. Unterstützung grundlegender REST Prinzipien

Abschnitt	Stufe	Kommentar
Ressourcenidentifikation und Ressourcenstruktur	2	Conventions-over-Configuration, Klassen bekommen ihr URI anhand der Struktur im Projekt. URI-Konfiguration auch explizit möglich
Ressourcentypen	2	Conventions-over-Configuration Ansatz ermöglicht viele Ressourcentypen
Hypermedia	0	
Medientypen	1	Standard Medientypen
Caching	2	Per Routes Datei konfigurierbar
Code-On-Demand	0	

Tabelle 4.9.: Auswertung Scooter Unterstützung grundlegender REST Prinzipien

Ressourcenidentifikation und Ressourcenstruktur

1 Scooter hat einen Conventions-over-Configuration Ansatz, das bedeutet Klassen werden Anhand ihrer Projektstruktur auf URI's gemapped, falls nicht explizit in der Routing Konfiguration eine andere gewählt wird.

2 Ja.

3 URI's werden auf Methoden von Controller-Klassen geleitet.

4 Nein.

Ressourcentypen

1 Es gibt die Singular-Resources [scoc] die dann mittels Kontrukten in der routes.properties verfeinert werden können.

2 Alle Resourrcen in Scooter sind in erster Linie Primärressourcen und können mittels Routing-Konfiguration abgeändert werden.

3 Ja, per Routing-Konfiguration können ganze Controller-Klassen/Ressourcen als Subressource deklariert werden.

4 Ja, per Routing-Konfiguration können reine Listenressourcen definiert werden.

5 Ja, per Routingkonfiguration können bspw. paginierte Repräsentationen generiert werden.

6 Nein.

7 Nein.

Hypermedia

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Medientypen

1 Ja, dazu werden ContentHandler Plugins benötigt. Diese implementieren eine *handle* Methode und werden in der Konfiguration registriert. Die *handle* Methode bekommt als Parameter den ganzen HTTP Request aus der sie dann eine HTTP Response generieren soll [scod].

2 Nein.

3 Die Content-Negotiation bei Scooter wird anhand der Routing-Konfiguration bewerkstelligt. Dabei wird die erste Deklaration in der routes.properties genommen die zur Request passt [scoe].

4 Die erste passende aus routes.properties die zum Request passt wird gewählt.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Caching

1 Es gibt WebCaching für das Cachen von Views und DataCaching für Caching zwischen Applikation und Datenbank.

2 Alle Caching relevanten Einstellungen können in einer Konfigurationsdatei festgelegt werden. Die können global und lokal pro Route festgelegt werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Code-On-Demand

1 Keine.

4.2.4. Erweiterte Technische Fähigkeiten

Abschnitt	Stufe	Kommentar
Protokollunterstützung jenseits von HTTP	0	
HTTP	2	HTTP im angemessenem Umfang nutzbar
Unterstützung für Transaktionen	1	JTA, JDBC
Security	1	Authentifikation
Asynchronität	0	
Zuverlässigkeit	0	
Umgang mit großen Daten	2	

Tabelle 4.10.: Auswertung Scooter Erweiterte Technische Fähigkeiten

Protokollunterstützung jenseits von HTTP

1 Keine.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

HTTP

1 Es ist möglich alle Verben zu nutzen, auch solche die nicht von der HTTP Spezifikation vorgegeben werden [scof].

2 Ja, mittels Controller Klassen lässt sich Semantik der Verben umgehen.

1 Ja, in ContentHandlern.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Unterstützung für Transaktionen

1 Ja, es ist möglich einen Request transaktional zu verarbeiten [scog].

2 Scooter ermöglicht es bereits bestehende Transkationen zu nutzen. Dazu wird der ImplicitTransactionManager genutzt [scog].

3 Scooter unterstütz aktuell JDBC, JTA und den TransaktionsManager des Containers für Transaktionen.

4 JTA wird unterstützt, somit auch 2PC.

Security

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Es ist möglich mit den mitgelieferten Werkzeugen, einen Login-Mechanismus zu generieren.

3 Ja, Scooter ist auf Standalone Applikationen ausgelegt.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Asynchronität

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Zuverlässigkeit

1 Nein.

2 Nein.

3 Nein.

Umgang mit großen Daten

1 Es gibt die Möglichkeit große Dateien ausserhalb des Hauptspeichers in einem Ordner zu speichern, falls eine gewisse Größe überschritten wird.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.3. VRaptor

Eckdaten

URL	http://vraptor.caelum.com.br/
Lizenz	Apache 2.0 Lizenz
Entwickler	Caelum (http://www.caelum.com.br/)

Tabelle 4.11.: Eckdaten VRaptor

Beschreibung Benutzt VRaptor (MVC framework von caelum). Benutzt Annotations um HTTP-Calls von Methoden beantworten zu lassen. (@Path mit URI pattern). Resourcen werden mit @Resource annotiert. Sieht der vorgehensweise in Jersey sehr ähnlich. Application-xml und Application-json wird unterstützt(benutzt XStream) Eigene Deserializer können implementiert werden für eigene Typen. Unterstützung für HyperMedia mittels Java-Interface "HypermediaResource".(Resourcen verlinkung)

Beispiel VRaptor bietet einen guten Ansatz, wie einzelne URIs die Ressource identifizieren. So ist es möglich ala JAX-RS mit der @Path Annotation die Methoden auf die Pfade zu mappen, außerdem bietet hier VRaptor noch die Möglichkeit HTTP Methoden direkt abzubilden(siehe 4.3). Möglichkeit besteht alle URIs zentral in einer Routing Klasse zu halten.4.4

Listing 4.3 VRaptor Ressourcen Beispiel

```
@Resource
public class ItemsController {
       @Get
       @Path("/items")
       public void list() {...}

       @Post("/client")
       public void add(Client client){...}

       @Delete("/client")
       public void remove(Client client) {...}
}
```

Listing 4.4 Zentrale Routing Klasse für URIs in dem Framework VRaptor

```
@Component
@ApplicationScoped
public class CustomRoutes implements RoutesConfiguration {
    public void config(Router router) {
        new Rules(router) {
            public void routes() {
                routeFor("/").is(ClientController.class).list();
                routeFor("/client/random").is(ClientController.class).random();
            }
        };
    }
}
```

4.3.1. Grundlagen

Abschnitt	Stufe	Kommentar
Allgemeines	2	Gute Dokumentation und Tutorials, erweiterbar, aktive Community
REST Server Applikationen	1	Auf Servlet ausgelegt, Standalone sehr aufwändig
REST Client Applikationen	2	Eigenes Clientprojekt Restfulie

Tabelle 4.12.: Auswertung VRaptor Grundlagen

Allgemeines

1 Ja, auf der offiziellen Dokumentationsseite existieren zwei Einführungshilfen, Codebeispiele für das Entwickeln mit den gegeben Annotationen und deren funktionsweise, bspw. Ressourcen. Integration in andere Technologien werden dort auch erörtert.

2 Es existieren Maillinglisten für Benutzer [vraa] und Entwickler [vrab], ausserdem wird VRaptor auf GitHub [vrac] zur Verfügung gestellt.

3 Apache 2.0 Lizenz.

4 Ja, es gibt zusätzlich zum Pluginmechanismus speziell ein Projekt auf GitHub, das für externe Plugins vorgesehen ist. [vrad] Die Entwickler von VRaptor nehmen brauchbare Plugins auch in das Release auf.

REST Server Applikationen

1 Ja [vrae].

2 Alle Umgebungen die die Servlet Spezifikation ab Version 2.5 unterstützen.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

REST Client Applikationen

1 Zur Client-Entwicklung in VRaptor wird das Projekt Restfulie vom gleichen Entwickler (Caelum) genutzt.

2 Restfulie besitzt einfache Beispiele zur Einführung, Erklärungen zum Nutzen von Links in Repräsentationen, Beispiele für Transitionen.

3 Es gibt Hilfsmittel zur Verarbeitung von Links, Automatische Serialisierung, Zustandsabhängige Aufrufe von Ressourcen.

Architektur und Funktionsweise

1 VRaptor ist ein MVC-Framework und bietet typisch Funktionen, Klassen für Model, View und Control an. Es werden erst Klassen für das Modell erstellt die dann Mittels Controllern eine View an die Clients schickt.

2 VRaptor baut auf dem Spring Framework und nutzt etwaige andere Technologien wie das Injection-Framework Google Guice. VRaptor benutzt zum Builden Apache Maven da kann man alle Abhängigkeiten einsehen. [vraf]

4.3.2. Entwicklung von REST basierten Anwendungen

Abschnitt	Stufe	Kommentar
Entwicklungsprozess/ Vorgehensmodell	0	
Modellierung von REST APIs	0	
Modellierungswerkzeuge	0	

Tabelle 4.13.: Auswertung VRaptor Entwicklung von REST basierten Anwendungen

Entwicklungsprozess/Vorgehensmodell

1 Nein.

2 Nein.

3 Da VRaptor ein MVC Framework ist lassen sich passende Entwicklungsprozesse auswählen.

Modellierung von REST APIs

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Nein.

3 Nein.

4 Nein.

Modellierungswerkzeuge

1 Nein.

2 Nein.

3 Nein.

4 Kann nicht beantwortet werden.

5 Nein.

6 Nein.

7 Nein.

8 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.3.3. Unterstützung grundlegender REST Prinzipien

Abschnitt	Stufe	Kommentar
Ressourcenidentifikation und Ressourcenstruktur	1	Annotationen ähnlich zu JAX-RS 1.1
Ressourcentypen	0	
Hypermedia	1	Spezielle Hypermedia-Ressource
Medientypen	1	Standard Medientypen, benutzt zentrale Medientypen Repository
Caching	1	ObservableResource Klasse, um Caching zu behandeln
Code-On-Demand	0	

Tabelle 4.14.: Auswertung VRaptor Unterstützung grundlegender REST Prinzipien

Ressourcenidentifikation und Ressourcenstruktur

1 Es werden Annotationen an Controller-Klassen-Methoden angehängt oder in einem DeploymenDeskriptor definiert.

2 Ja.

3 URI werden anhand der Annotationen (bzw. definierten URIs im Deployment Deskriptor) auf Controller-Klassen-Methoden weitergereicht. Man kann auch zusätzlich Router-Klassen verwenden. [vrag]

4 Nein.

Ressourcentypen

1 Primärresourcen.

2 Mit der @Resource Annotation an Controller-Klassen können Primärressourcen realisiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Hypermedia

1 Es gibt die Hypermedia-Resource, diese kann abhängig von ihrem Zustand Relationen an den Client schicken.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Medientypen

1 VRaptor unterstütz alle gängigen XML, JSON Medientypen. Leider konnte keine genaue List ermittelt werden.

2 VRaptor und Restfulie benutzen Medie, eine Medientyp-Repository mit der man eigene Serialisierer und Deserialisierer registrieren kann. [https://github.com/caelum/medie]

3 Ja.

4 Ja.

5 Falls der Restfulie-Client keinen passenden Medientyp vom Server kriegt wählt er, falls vorhanden einen den er kennt automatisch.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

8 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Caching

1 Es gibt die Möglichkeit eine ObservableResource zu nutzen um Caching Funktionen direkt in den Ressource-Klassen zu behandeln. [vrah]

2 Nein.

3 Nur das Validierungsmodell.

4 Ja.

5 Nein.

Code-On-Demand

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.3.4. Erweiterte Technische Fähigkeiten

Abschnitt	Stufe	Kommentar
Protokollunterstützung jenseits von HTTP	0	
HTTP	2	HTTP im angemessenem Umfang nutzbar
Unterstützung für Transaktionen	1	Spring, JPA
Security	2	SSL und Plugins für Authentifizierung
Asynchronität	1	Non-Blocking Requestverarbeitung
Zuverlässigkeit	0	
Umgang mit großen Daten	2	

Tabelle 4.15.: Auswertung VRaptor Erweiterte Technische Fähigkeiten

Protokollunterstützung jenseits von HTTP

1 Keine.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

HTTP

1 Ja.

2 Da HTTP-Aufrufe nur auf Java-Methoden ümgeleitet"werden, bleibt die Einhaltung der Verbsemantik in Bezug auf Idempotenz und Sicherheit in der Verantwortung des Entwicklers.

1 Ja.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Unterstützung für Transaktionen

1 Ja, VRaptor benutzt dazu Annotationen. Darunter werden Transaktions-Funktionen von Spring, JPA genutzt.

2 Nein.

3 Ja.

4 Nicht explizit in VRaptor selbst.

Security

1 Ja, man kann eine Applikation so konfigurieren das SSL verwendet wird. [vrai]

2 Ja, es gibt Plugins die Authentifizierung unterstützen. [vraj] [vrak]

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Asynchronität

1 Die Request und Responses sind in VRaptor von Servlet HttpServletRequest abgeleitete Klassen und erlauben die darin enthaltenen Funktionen für asynchrone Programmiermodelle.

2 Nein.

Zuverlässigkeit

1 Gar nicht.

2 Nein.

3 Nein.

Umgang mit großen Daten

1 Ja.

2 Es gibt die Möglichkeit falls zu große Dateien auftreten mittels eines Validators [vral] das abzufangen und selbst geschriebener Logik diese zu behandeln. [vram]

4.4. Resthub

Eckdaten

URL	http://resthub.org/
Lizenz	Apache License 2.0
Entwickler	

Tabelle 4.16.: Eckdaten Resthub

Beschreibung RestHub ist ein Stack von Spring Framework für die Serverseite und die Backbone.js für die Clientseite. RestHub wurde auf HTML5 Applications ausgelegt. Die Kommunikation funktioniert über die Websockets oder REST webservices. RESThub definiert eigene Profile, welche dann ausgeführt werden. Man kann auch die Profile von Spring verwenden.

Listing 4.5 URI Builder des Spring Stacks des REST Hub Frameworks

```
UriComponents uriComponents =
       UriComponentsBuilder
       .fromUriString("http://example.com/hotels/{hotel}/bookings/{booking}")
       .build();

URI uri = uriComponents.expand("42", "21").encode().toUri();
```

Beispiel Um die Serverseitige Unterstützung für RESTful Applikationenzu ermöglichen, wurden Annotationen in das bestehende MVC web framework hinzugefügt. Mit der Annotation @Controller lassen sich RESTful Webseiten und Applikationen mit Hilfe von @PathVariable realisieren. Mit UriComponentsBuilder und UriComponents können URLs dekodiert und kodiert werden. (Siehe 4.5)

Clientseitige Unterstützung über das RestTemplate, welches sich an die Klassen wie JdbcTemplate und JmsTemplate anlehnen. Server und Client benutzen beide die HttpConverter, um den Austausch von Requests und Responses durchführen zu können.

4.4.1. Grundlagen

Abschnitt	Stufe	Kommentar
Allgemeines	1	Gute Dokumentation
REST Server Applikationen	1	Nur Servlet möglich
REST Client Applikationen	1	Auf Browser-Clients ausgelegt. Eine große Bibliothek Backbone.js wird bereitgestellt.

Tabelle 4.17.: Auswertung Resthub Grundlagen

Allgemeines

1 RestHub bietet eine gute Dokumentation für den Spring Stack, sowie für den Backbone.js Stack.

2 Es existiert ein Forum bei Google eine Gruppe:"resthub-dev", weitere Foren, Mailinglisten oder ähnliches sind nicht bekannt.

3 Spring Framework ist unter Apache 2.0 lizensiert. Backbone.js ist unter MIT License lizensiert.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

REST Server Applikationen

1 Nein.

2 In einem Servlet Container kann eine WAR Datei ausgeführt werden. Als Empfehlung wird heir Jetty Servlet Engine angegeben.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

REST Client Applikationen

1 Ja, das Framework bietet mit dem Spring Stack einen Web Client an. Backbone.js bietet jedoch viel mächtigere Scripte zur Entwicklung Browser gestützten Clients.

2 Es existiert eine vorgefertigte Klasse, welche sofort in Gebrauch genommen werden kann. Hierbei unterscheidet man zwischen synchronen und asynchronen Aufrufen.

3 Mit Hilfe von Jackson ist es möglich, sofort JSON Objekte in Klassen zu konvertieren.

Architektur und Funktionsweise

1 In einer Routing Datei wird festgelegt, wie der Server bei Aufrufen verhalten soll. So kann ein GET /item weiter an eine Controllermethode geleitet werden, ähnlich dem vorgehen im Play Framework.

2 Um eine Applikation ausführen zu können benötigt man einen Servlet Container.

4.4.2. Entwicklung von REST basierten Anwendungen

Abschnitt	Stufe	Kommentar
Entwicklungsprozess/ Vorgehensmodell	0	
Modellierung von REST APIs	0	
Modellierungswerkzeuge	0	

Tabelle 4.18.: Auswertung Resthub Entwicklung von REST basierten Anwendungen

Entwicklungsprozess/Vorgehensmodell

1 Nein.

2 Es existieren keine speziellen Mechanismen, welche ein Vorgehensmodell bevorteilen.

3 Der typische Entwicklungsprozess sieht folgender maßen aus: Man beginnt mit dem Entwurf der Ressourcen und deren URI Struktur. Ab dann werden die Controllerfunktionalitäten implementiert.

Modellierung von REST APIs

1 Spring unterstützt keine automatische WADL Generierung.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Modellierungswerkzeuge

1 Nein.

2 Nein.

3 Nein.

4 Kann nicht beantwortet werden.

5 Nein.

6 Nein.

7 Nein.

8 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.4.3. Unterstützung grundlegender REST Prinzipien

Abschnitt	Stufe	Kommentar
Ressourcenidentifikation und Ressourcenstruktur	1	Zentrale Routingdatei
Ressourcentypen	0	
Hypermedia	0	
Medientypen	1	Standard Medientypen
Caching	2	Per Annotation konfigurierbar
Code-On-Demand	0	

Tabelle 4.19.: Auswertung Resthub Unterstützung grundlegender REST Prinzipien

Ressourcenidentifikation und Ressourcenstruktur

1 Es gibt eine gemeinsame Routing Datei in dieser werden alle URI gesammelt und auf Controller Methoden gemappt. Variablen in Pfaden können benutzt werden. Diese können auch über regex Regeln festgesetzt werden.

2 Siehe Frage 1.

3 s.o.

4 Nein.

Ressourcentypen

1 Vordefinierte Ressourcen Typen gibt es nicht. Es besteht aber eine Möglichkeit mit Hilfe von Jackson und JSON, ein schnelles erstellen von Ressourcen aus einem JSON Objekt.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Wie bereits erwähnt mit Jackson und JSON.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Hypermedia

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Medientypen

1 Unterstützt werden die Medientypen, die in der Klasse org.springframework.http.MediaType definiert sind.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Client-side Content-Negotiation wird über das setzen von Accept-Headern unterstützt.

4 Server Side Content-Negotiation lässt sich auf zwei Arten implementieren. Man kann hierfür von einer Java Klasse WebMvcConfigurerAdapter erben und somit Zugang zu der Methode configureContentNegotiation() zu erlangen. Indieser können Medientypen festgesetzt werden. Eine weitere Möglichkeit bietet sich über eine XML Konfigurationsdatei an.

5 Entweder erfolgt dies über die Endung der URI beispielsweise .json, .xml (Funktionalität ist dann ähnlich Dateiendungen), oder über Reihenfolge, in der Medientypen auf dem Server gesetzt sind.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Caching

1 Das Caching wird von dem Spring Stack auf der Server side unterstüzt.

2 Es gibt eine Möglichkeit die Controllermethoden mit einer Annotation @Cacheable zu definieren. Hier kann sowohl Parameter, als auch Rückgabe der Methode als @Cacheable markiert werden.

3 Es existiert das Conditional Caching.

4 Ja.

5 Ja.

Code-On-Demand

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.4.4. Erweiterte Technische Fähigkeiten

Abschnitt	Stufe	Kommentar
Protokollunterstützung jenseits von HTTP	0	
HTTP	2	HTTP im angemessenem Umfang nutzbar
Unterstützung für Transaktionen	2	Spring
Security	1	SSL
Asynchronität	2	Non-Blocking
Zuverlässigkeit	0	
Umgang mit großen Daten	1	Per Stream

Tabelle 4.20.: Auswertung Resthub Erweiterte Technische Fähigkeiten

Protokollunterstützung jenseits von HTTP

1 Keine.

2 Kann nicht beantwortet werden.

3 Nein.

4 Kann nicht beantwortet werden.

5 Module und Converter können dabei Helfen neue Protokolle einzubinden.

HTTP

1 DELETE, GET, HEAD, OPTIONS, POST, PUT, TRACE, Patch werden unterstützt.

2 Da HTTP-Aufrufe nur auf Java-Methoden ümgeleitet"werden, bleibt die Einhaltung der Verbsemantik in Bezug auf Idempotenz und Sicherheit in der Verantwortung des Entwicklers.

1 Es besteht die Möglichkeit alle Header mit Hilfe der Klasse org.springframework.http.HttpHeaders zu setzen.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Unterstützung für Transaktionen

1 Es werden Transaktionen unterstützt, dabei werden die vom Spring Framework angebotenen Fähigkeiten genutzt. Dabei muss das Spring Interface *PlatformTransactionManager* implementiert werden.

2 Ja.

3 Resthub ist ein vorgefertigter Stack, aus Spring und Backbone.js, somit können alle Features von Spring benutzt werden um externe Transaktionstechniken zu intergrieren.

4 Spring unterstützt 2PC, somit auch Resthub.

Security

1 SSL wird unterstützt. Dabei kann hier die JSSE oder OpenSSL genutzt werden.

2 Kann nicht beantwortet werden.

3 Kann nicht beantwortet werden.

4 Kann nicht beantwortet werden.

5 Kann nicht beantwortet werden.

Asynchronität

1 Mit einer Annotation @Async ist es möglich eine Methode asynchron auszuführen.

2 Mit Backbone.js ist es möglich einen Callback zu registrieren, so dass hier ein asynchroner Aufruf möglich ist.

Zuverlässigkeit

1 Gar nicht.

2 Nein.

3 Nein.

Umgang mit großen Daten

1 Es besteht die Möglichkeit ein Stream bereit zu stellen. Sonst sind keine weiteren Mechanismen bekannt.

2 Kann nicht beantwortet werden.

4.5. Apache CXF

Eckdaten

URL	http://cxf.apache.org/
Lizenz	Apache 2.0 Lizenz
Entwickler	Apache Software Foundation

Tabelle 4.21.: Eckdaten Apache CXF

Beschreibung Apache CXF ist ein Open-Source Service Framework, um Services mithilfe von API's/Spezifikationen, wie JAX-WS und JAX-RS, zu implementieren. CXF hat eine Reihe von unterstützten Protokollen wie SOAP, XML/HTTP, RESTful HTTP oder CORBA die über eine Reihen von Transportprotokollen arbeiten wie HTTP, JMS oder JBI. [cxfa]

Beispiel Da CXF JAX-RS 1.1 implementiert, werden Services entsprechend der Spezifikation entwickelt. Ein Beispiel 4.1 und eine genauere Beschreibung kann bei Jersey 4.1 eingesehen werden.

4.5.1. Grundlagen

Abschnitt	Stufe	Kommentar
Allgemeines	2	Ausführliche Dokumentation, aktive Community, explizit auf Erweiterungen ausgelegt
REST Server Applikationen	2	Servlet und Standalone komfortabel möglich, OSGi möglich
REST Client Applikationen	2	Verschiedene Clients mit unterschiedlichen Funktionalitäten für verschiedene Ansprüche

Tabelle 4.22.: Auswertung Apache CXF Grundlagen

Allgemeines

1 Es gibt einen umfassenden User Guide um das generelle arbeiten mit CXF zu erleichtern. Für REST Entwicklung gibt es drei Unterkategorien Entwicklung mit JAX-RS 1.1 [cxfb], JAX-WS Provider [cxfc] und HTTPBinding [cxfd] [cxfe].

2 Es gibt mehrere Maillinglisten [cxff] und einen IRC-Channel **#cxf@irc.codehaus.org**.

3 Apache Lizenz 2.0.

4 Ja, CXF hat explizit möglichkeiten zur Erweiterung [cxfg].

REST Server Applikationen

1 Ja [cxfh].

2 CXF kann auf verschiedenen Application Servern deployed werden [cxfi]. Es gibt Möglichkeiten CXF in einem OSGI Framework zu betreiben Da CXF standalone betrieben werden kann, läuft CXF auf jedem System das eine JVM zur Verfügung stellt.

3 Für verschiedene Umgebungen müssen teils andere Konfigurationen gewählt werden, die im Deployment Deskriptor eingestellt werden.

REST Client Applikationen

1 Ja.

2 Es gibt drei Arten von REST-Clients: Proxy-based, HTTP-centric und XML-centric. Der Proxy-based Client erlaubt es die im Server entwickelten JAX-RS Resource Klassen wieder zuverwenden. Somit kann direkt auf Java-Klassen gearbeitet werden. Der HTTP-centric Client stellt bekannte Konstrukte aus HTTP zur verfügung. So können HTTP Header gesetzt werden und HTTP Verben explizit genutzt werden. XML-centric Clients erlauben Ressourcen Repräsentationen aus dem XML Umfeld mittels XPath Queries und XSLT Stylesheets zu verarbeiten.

3 Alle drei Clients erlauben Repräsentationen direkt in vorgesehene Java-Objekte zu (de-)serialisieren. Es ist möglich einen Proxy-based Client in einen HTTP-centric umzuwandeln und umgekehrt. Es können zur Laufzeit Clients generiert werden die es erlauben per Java-Relflection diese bestimmten Klassen entsprechend anzupassen.

Architektur und Funktionsweise

1 CXF teilt sich in 7 Komponenten auf. Eine Komponente ist der Bus, dieser ist ein Regsiter für Erweiterungen, Interceptoren (Auffänger) und Properties. Dieser gilt als der "Backbone" von CXF. [cxfg] Die Front-end Komponente bietet ein Programmiermodell an, mit dem Services erstellt werden können. In dieser ist auch die JAX-RS 1.1 implementierung enthalten. [cxfj] *Messaging & Interceptors* Komponente besteht aus Messages, Interceptoren und Interceptor-Ketten. Dabei ist die Idee das Messages durch Interceptor-Ketten geschickt werden die diese verarbeiten.Beispielweise ist ein Interceptor innerhalb der Kette für das konkrete XML parsen oder Verschlüsselung zuständig.[cxfk] Das Service-Model ist eine Komponente die in CXF angelegte Services beschreibt, dabei wurde das Model an WSDL angelehnt. [cxfl] Die Databinding Komponente besteht aus Konvertern die aus gegebenen Formaten diesen in XML transformieren und umgekehrt, da aber CXF JAX-RS 1.1 implementiert lassen sich auch JAX-RS Provider zur Konvertierung von Medientypen nutzen.[cxfm] [cxfn] Die Protocol Bindings Komponente wird verwendet um Messages auf konkrete Transport Protokolle zu transformieren. [cxfo] Die Komponente Transports ist für das Senden und Empfangen innerhalb verschiedener Protokolle zuständig.[cxfp]

2 Apache CXF erlaubt Spring integration und viele weitere Möglichkeiten externe Technologien zu integrieren/verwenden, ist aber selbst nicht davon abhängig.

4.5.2. Entwicklung von REST basierten Anwendungen

Abschnitt	Stufe	Kommentar
Entwicklungsprozess/ Vorgehensmodell	0	
Modellierung von REST APIs	2	erlaubt von Code zu WADL und WADL zu Code Generierung
Modellierungswerkzeuge	0	

Tabelle 4.23.: Auswertung Apache CXF Entwicklung von REST basierten Anwendungen

Entwicklungsprozess/Vorgehensmodell

1 Nein.

2 Nein.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Modellierung von REST APIs

1 Es können WADLs aus einer laufenden Applikation generiert werden, zusätzlich gibt es das Tool wadl2java um Java-Code aus einer WADL zu erstellen.[cxfq]

2 Nein.

3 Ja.

4 Ja, aus der WADL lässt sich Client und Server Code generieren.

Modellierungswerkzeuge

1 Nein.

2 Nein.

3 Nein.

4 Kann nicht beantwortet werden.

5 Nein.

6 Nein.

7 Nein.

8 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.5.3. Unterstützung grundlegender REST Prinzipien

Abschnitt	Stufe	Kommentar
Ressourcenidentifikation und Ressourcenstruktur	1	JAX-RS 1.1 Annotationen
Ressourcentypen	0	
Hypermedia	0	
Medientypen	1	JAX-RS 1.1
Caching	1	Caching durch Cachingframework
Code-On-Demand	0	

Tabelle 4.24.: Auswertung Apache CXF Unterstützung grundlegender REST Prinzipien

Ressourcenidentifikation und Ressourcenstruktur

1 Gemäß JAX-RS erfolgt die Identifikation über URI-Elemente, die per @Path Annotation an Klassen und Methoden gebunden werden können. Dabei können sowohl einzelne URI-Elemente, als auch Teilpfade einer URI verwendet werden. Es ist damit möglich Aufruf-Hierarchien über verschiedene Klassen und Methoden hinweg zu realisieren.

2 Ja.

3 Siehe dazu JAX-RS 1.1 Spezifikation. [jaxa, JAX-RS 1.1 Abschnitt 3.7]
Zusätzlich ist es möglich die Wahl der Klassen und/oder der Methoden zu beeinflussen. [cxfr]

4 Nein.

Ressourcentypen

1 Primärressourcen und Subressourcen können mit Mitteln aus JAX-RS 1.1 unkompliziert realisiert werden, sind aber nicht speziell als solche deklariert.

2 Gemäß JAX-RS 1.1 können Klassen mittels @Path Annotationen als Primärressource verfügbar gemacht werden.

3 Gemäß JAX-RS 1.1 können Methoden innerhalb von Klassen mittels @Path Annotationen als Subressourcenidentifikator genutzt werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Hypermedia

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Medientypen

1 Alle aus der JAX-RS Spezifikation.

2 CXF implementiert JAX-RS, somit können Provider implementiert werden die das Serialisieren/Deserialisieren von Medientypen übernehmen. Dafür muss ein Interface implementiert werden (*MessageBodyReader* bzw. *MessageBodyWriter*) und mit einer *@Provider* Annotation versehen werden.

3 Ja, wird gemäß der JAX-RS Spezifikation durchgeführt.

4 Ja, wird gemäß der JAX-RS Spezifikation durchgeführt.

5 Wird gemäß der JAX-RS Spezifikation durchgeführt.

6 Wird gemäß der JAX-RS Spezifikation durchgeführt.

7 Wird gemäß der JAX-RS Spezifikation durchgeführt.

8 Wird gemäß der JAX-RS Spezifikation durchgeführt.

Caching

1 Es ist mögliche CachingFrameworks wie bspw. Ehcache-Web [ehca] durch Konfiguration in der web.xml zu nutzen.

2 Das benutzte CachingFramework muss komplett selbst konfiguriert werden.

3 Kommt auf das eingesetzte Framework an.

4 E-Tags werden unterstützt, müssen aber vom Entwickler selbst definiert werden.

5 Nein.

Code-On-Demand

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.5.4. Erweiterte Technische Fähigkeiten

Abschnitt	Stufe	Kommentar
Protokollunterstützung jenseits von HTTP	2	JMS
HTTP	2	JAX-RS 1.1
Unterstützung für Transaktionen	0	
Security	2	SSL,Keystore, Signaturen,viele Authentifizierungen XML-Verschlüsselung und XML-Signaturen
Asynchronität	1	Unterstützung für Apache HttpAsyncClient (kann Non-Blocking I/O)
Zuverlässigkeit	0	
Umgang mit großen Daten	2	Auslagerung auf Festplatte möglich

Tabelle 4.25.: Auswertung Apache CXF Erweiterte Technische Fähigkeiten

Protokollunterstützung jenseits von HTTP

1 JMS [cxfs].

2 Es müssen Konfigurationsdateien erstellt werden um Queue oder Topics zu konfigurieren [cxft].

3 Ja.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Ja, innerhalb der Protocol Binding und Transports Komponenten.

HTTP

1 JAX-RS 1.1 definiert als Annotationen GET, PUT, POST, DELETE und HEAD. Desweiteren verlangt JAX-RS 1.1 auch noch Antworten auf OPTIONS Anfragen. Weiterhin können über die Annotation HttpMethod weitere HTTP-Methoden gemapped werden.

2 Da die HTTP-Verb-Annotationen lediglich zur Auswahl von Java-Methoden verwendet werden, bleibt die Einhaltung der Verbsemantik in Bezug auf Idempotenz und Sicherheit in der Verantwortung des Entwicklers.

1 Ja.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Unterstützung für Transaktionen

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Kann nicht beantwortet werden.

3 Ja, JTA kann genutzt werden über JMS Konfiguration.

4 Durch JTA ist es auch möglich 2PC zu nutzen.

Security

1 Der client von Apache CXF benutzt automatisch Zertifikate und Keystores, die Teil der JDK sind, wenn eine url mit “https” aufgerufen wird. Nur bei “custom” und selbst-signierten Zertifikaten kann eine Konfiguration des keystores und des trust managers von Nöten sein. Der Apache CXF standalon HTTP transport kann mittels Konfiguration SSL-fähig gemacht werden.

2 CXF kann Standalone betrieben werden und ist mittels Konfigurationsdatei voll konfigurierbar.

3 Client beherrscht Basic und Digest Access, darüber hinaus kann noch dynamic authorization, NTLM Authentication und Spnego Authentication genutzt werden. Der Server beherrscht BasicAuth. [cxfu]

4 Es gibt automatische XML-Verschlüsselung um die MessageEntity zu verschlüsseln.

5 XML-Signaturen auf beiden Seiten.

Asynchronität

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Apache CXF bietet unter anderem die Apache HTTP Components HttpAsyncClient library. [apaa]

Zuverlässigkeit

1 Für idempotente Methoden ermöglicht CXF Clients andere Server zu nutzen falls der Primäre nicht erreichbar ist mittels Failover-Strategien. [cxfv] Weitere Probleme müssen vom Entwickler selbst gelöst werden.

2 Nein.

3 Nein.

Umgang mit großen Daten

1 Ja, es ist möglich Speicher auf der Festplatte des Systems zu nutzen um große Dateien darin auszulagern. [cxfw]

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.6. Resteasy

Eckdaten

URL	http://www.jboss.org/resteasy/
Lizenz	LGPL v3.
Entwickler	JBoss

Tabelle 4.26.: Eckdaten Resteasy

Listing 4.6 Ein Beispiel einer mit Resteasy entwickelten Methode.

```
@Path("/events")
@RequestScoped
public class EventService {
	@Inject
	private EntityManager em;

	@GET
	@Produces(MediaType.APPLICATION_JSON)
	public List<Event> getAllEvents() {
		final List<Event> results =
			em.createQuery(
			"select e from Event e order by e.name").getResultList();
		return results;
	}
}
```

Beschreibung Resteasy ist eine zertifizierte JAX-RS Implementierung. Zusätzlich bietet das Framework die Funktionalitäten für die Client Entwicklung, Caching, Asynchrone Programmiermodelle und JPA Support. Resteasy kann in jedem Servlet Container ausgeführt werden. Doch die starke Bindung an den JBoss Application Server sorgt für eine angenehmere Ausführung der Applikationen in dieser Umgebung.

Beispiel In JAX-RS wird vieles über Annotationen deklariert. Dabei werden die URIs mittels der @Path Annotation an die Methoden gebunden. Siehe 4.6.

4.6.1. Grundlagen

Abschnitt	Stufe	Kommentar
Allgemeines	2	Ausführliche Dokumentation, aktive Community
REST Server Applikationen	1	Servlet Container
REST Client Applikationen	2	Client Framework wird angeboten, Server und Client können gemeinsame Interfaces verwenden

Tabelle 4.27.: Auswertung RESTEasy Grundlagen

Allgemeines

1 Eine Dokumentation ist auf der offiziellen Seite von Resteasy zu finden. In dieser Dokumentation werden unter anderem einige Beispiele vorgestellt.

2 Es existieren eine offizielle Wiki, Mailing Listen, sowie Bücher zu Resteasy.

3 Resteasy steht unter der ASL 2.0 Lizenz. Es werden keine Bibliotheken von Drittanbietern verbreitet, welche unter der GPL Lizenz lizensiert sind. Es werden jedoch Bibliotheken von Drittanbietern in Resteasy mitverbreitet, welche unter der ASL 2.0 und LGPL Lizenzen lizensiert sind.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

REST Server Applikationen

1 Es ist möglich eine Standalone Applikation zu realisieren. Dafür sind nötige Schritte durchzuführen. Das Vorgehen ist in der Dokumentation beschrieben. [resa]

2 Die Applikation kann in einem JBoss 7 Application Server ausgeführt werden.

3 Mit Hilfe von verschiedenen Konfigurationen lässt sich die Anwendung auf weitere Servlet Container portieren.

REST Client Applikationen

1 Resteasy bietet mit dem Client Framework eine Möglichkeit einen Clientapplikation zu entwickeln. Dieses Client Framework basiert auf dem Apache HttpClient.

2 Die Dokumentation liefert nötige Informationen zur Cliententwicklung.

3 Es gibt die Möglichkeit die für Server-seite entwickelten MessageBodyWriter und Reader auch für den Client verfügbar zumachen, darüber hinnaus kann man Filter entwickeln.

Architektur und Funktionsweise

1 Die Funktionsweise des Frameworks gleicht den weiteren Frameworks, welche die JAX-RS Spezifikation implementieren. Mittels @Path und den HTTP Verben Annotationen werden die Methoden deklariert.

2 Als Applikation Server wird zum Ausführen JBoss AS 7 eingesetzt. Die Applikation kann jedoch auch auf anderen Servlet Container ausgeführt werden.

4.6.2. Entwicklung von REST basierten Anwendungen

Abschnitt	Stufe	Kommentar
Entwicklungsprozess/ Vorgehensmodell	o	
Modellierung von REST APIs	o	
Modellierungswerkzeuge	o	

Tabelle 4.28.: Auswertung RESTEasy Entwicklung von REST basierten Anwendungen

Entwicklungsprozess/Vorgehensmodell

1 Es wird kein Vorgehensmodell gefordert.

2 Nein.

3 Um später weniger Korrekturen an der Applikation durchführen zu müssen, sollten zunächst die Ressourcen, die URI Struktur und ein geeignetes Modell zu implementieren. Dann kann die Geschäftslogik implementiert werden.

Modellierung von REST APIs

1 Nein.

2 Nein.

3 Nein.

4 Nein.

Modellierungswerkzeuge

1 Nein.

2 Nein.

3 Nein.

4 Kann nicht beantwortet werden.

5 Nein.

6 Nein.

7 Nein.

8 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.6.3. Unterstützung grundlegender REST Prinzipien

Abschnitt	Stufe	Kommentar
Ressourcenidentifikation und Ressourcenstruktur	1	JAX-RS 1.1 Annotationen
Ressourcentypen	0	
Hypermedia	2	Atom Links, LinkHeader, Zustandsabhängig Links
Medientypen	1	JAX-RS 1.1 und weitere
Caching	2	Per Annotation konfigurierbar
Code-On-Demand	0	

Tabelle 4.29.: Auswertung RESTEasy Unterstützung grundlegender REST Prinzipien

Ressourcenidentifikation und Ressourcenstruktur

1 Gemäß JAX-RS erfolgt die Identifikation über URI-Elemente, die per @Path Annotation an Klassen und Methoden gebunden werden können. Dabei können sowohl einzelne URI-Elemente, als auch Teilpfade einer URI verwendet werden. Es ist damit möglich Aufruf-Hierarchien über verschiedene Klassen und Methoden hinweg zu realisieren.

2 Ja. Resteasy bietet die Möglichkeit in der @Path Annotation Variablen zu deklarieren. Diese können dann über weitere Annotationen genutzt werden.

3 Siehe dazu JAX-RS 1.1 Spezifikation. [jaxa, JAX-RS 1.1 Abschnitt 3.7]

4 Nein.

Ressourcentypen

1 Primärressourcen und Subressourcen können mit Mitteln aus JAX-RS 1.1 unkompliziert realisiert werden, sind aber nicht speziell als solche deklariert.

2 Gemäß JAX-RS 1.1 können Klassen mittels @Path Annotationen als Primärressource verfügbar gemacht werden.

3 Gemäß JAX-RS 1.1 können Methoden innerhalb von Klassen mittels @Path Annotationen als Subressourcenidentifikator genutzt werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Hypermedia

1 Ja, es können Bedingungen definiert werden um Links zu Repräsentationen hinzuzufügen wenn diese erfüllt ist.

2 RESTEasy implementiert Atom Links und die Link Header Spezifikation, es können somit Links im HTTP Header angegeben werden.

3 Ja, es können Annotationen verwendet werden, die es ermöglichen JAX-RS Ressourceklassen anzugeben um diese dann mittels einem Links verfügbar zu machen.

4 Es werden bezogen auf die JAX-RS Annotation einer Methode, default-Werte aus dem jeweiligen Standard (Atom, Linkheader) verwendet. Bspw. wird für eine @GET annotierte Methode die eine Liste zurückgibt als Relation *collection* gewählt.

5 Es können einfache Strings für die Typisierung verwendet werden.

6 Atom Links und LinkHeader.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Medientypen

1 Alle MedienTypen der JAX-RS Spezifikation und zusätzlich **atom+*** . [resb]

2 Es ist möglich eigene Medientypen zu definieren. Dazu müssen MessageBodyReader und Writer definiert werden, die für den eigenen Medientyp verantwortlich sind.

3 Ja.

4 Mit @Produces und @Consumes Annotation wird Server-side Content-Negotiation beeinflusst.

5 Die Medientypen können in einer web.xml definiert werden. Das Feld resteasy.media.type.mappings muss dabei mit den Medientypen gesetzt werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

8 Die Medientypen können in der web.xml definiert werden. Das Feld resteasy.language.mappings muss dabei mit den Medientypen gesetzt werden. Accept Header können gesetzt werden. Die weitere Möglichkeit besteht in der Query den Medientypen anzugeben.

Caching

1 Resteasy unterstüzt sowohl Server-side Caching, als auch Client-side Caching. Mit den Annotationen @Chache und @NoCache wird den Methoden mitgeteilt, ob der erfolgreiche Aufruf gecachet werden soll, oder nicht. Dabei muss man beachten, dass dies nur auf Methoden mit dem @GET HTTP Verb zutrifft. Alle anderen HTTP Verben werden nicht unterstützt. Auf der Client Seite wird der Browser Cache verwendet, sofern die Benutzung erlaubt ist.

2 Es können default-Werte für die @Cache Annotation gesetzt werden. Die @Cache Annotation muss jedoch auf den Methoden genutzt werden.

3 Ja.

4 Ja.

5 Ja.

Code-On-Demand

1 Nein.

4.6.4. Erweiterte Technische Fähigkeiten

Abschnitt	Stufe	Kommentar
Protokollunterstützung jenseits von HTTP	0	
HTTP	2	JAX-RS 1.1
Unterstützung für Transaktionen	0	
Security	2	SSL, OAuth, Signaturen, Verschlüsselung
Asynchronität	2	Non-Blocking-IO, Asynchrone Requests
Zuverlässigkeit	0	
Umgang mit großen Daten	2	

Tabelle 4.30.: Auswertung RESTEasy Erweiterte Technische Fähigkeiten

Protokollunterstützung jenseits von HTTP

1 Es ist nur HTTP möglich.

2 Kann nicht beantwortet werden.

3 Nein.

4 Ja.

5 Kann nicht beantwortet werden.

HTTP

1 JAX-RS 1.1 definiert als Annotationen GET, PUT, POST, DELETE und HEAD. Desweiteren verlangt JAX-RS 1.1 auch noch Antworten auf OPTIONS Anfragen. Weiterhin können über die Annotation HttpMethod weitere HTTP-Methoden gemapped werden.

2 Da die HTTP-Verb-Annotationen lediglich zur Auswahl von Java-Methoden verwendet werden, bleibt die Einhaltung der Verbsemantik in Bezug auf Idempotenz und Sicherheit in der Verantwortung des Entwicklers.

1 Ja.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Unterstützung für Transaktionen

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Kann nicht beantwortet werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Security

1 SSL, OAuth von Haus aus nutzbar.

2 OAuth.

3 Nein.

4 Ja.

5 Ja.

Asynchronität

1 Das Verarbeiten eines Requests kann asynchron stattfinden, dies muss zuerst im Servlet Container, wie Tomcat oder JBoss eingestellt werden. Die Dokumentation von Resteasy liefert hier einige Beispiele.

2 Ja.

Zuverlässigkeit

1 Gar nicht.

2 Nein.

3 Nein.

Umgang mit großen Daten

1 Ja. Resteasy ermöglicht den Austausch großer Dateien mit Hilfe der MultiPartProvider Klasse.

2 siehe 1.

4.7. Wink

Eckdaten

URL	http://wink.apache.org/
Lizenz	Apache 2.0 Lizenz
Entwickler	Apache Software Foundation

Tabelle 4.31.: Eckdaten Wink

Beschreibung Apache Wink betreibt zwei Module, den Wink Server und Wink Client.Der Wink Server implementiert JAX-RS 1.1 und definiert darüber noch eigene Features die dem REST-Prinzip helfen wollen.Der Wink Client baut auf JDK HttpUrlConnection auf und erweitert somit diesen für die entwicklung.[win]

Beispiel Da Wink JAX-RS 1.1 implementiert, werden Services entsprechend der Spezifikation entwickelt. Ein Beispiel 4.1 und eine genauere Beschreibung kann bei Jersey 4.1 eingesehen werden.

4.7.1. Grundlagen

Abschnitt	Stufe	Kommentar
Allgemeines	1	Dokumentation vorhanden, auf Erweiterung ausgelegt
REST Server Applikationen	1	Auf Servlet ausgelegt
REST Client Applikationen	1	Client-API mit Standardfunktionalität vorhanden

Tabelle 4.32.: Auswertung Apache Wink Grundlagen

Allgemeines

1 Es gibt ein Wiki und Dokumentation zu allen Releases bis 1.2.1 (aktuelle Version ist 1.3.0). Beim Teil „Getting Started“sind allerdings fast alle Kategorien noch TBD (to be done).

2 Es gibt mehrere Mailinglisten und einen (leeren) IRC-Channel.

3 Apache License, version 2.0.

4 Ja, Apache Wink hält sogar ein Verzeichnis dafür bereit.

REST Server Applikationen

1 Apache Wink ist darauf ausgelegt als Komponente zusammen mit anderen Servern bspw. Apache Geronimo oder Apache Tomcat zu laufen.

2 Apache Wink kann als .war Datei auf allen Servern ausgeführt werden, die .war Dateien unterstützen.

3 Als .war Distribution ist Apache Wink in allen Umgebungen lauffähig, die es erlauben eine .war File zu deployen.

REST Client Applikationen

1 Ja.

2 . Highlevel-HTTP-Aufrufe, (De-)Serialisierung von Objekten.

3 Der HTTP-Aufrufe, das Setzen der wichtigsten Header ist mittels Java-Methoden auf einem Objekt der Klasse Resource möglich. (De-)Serialisierung von Objekten ist JAX-RS typisch über MessageBodyReader und -Writer möglich. Die Standard Medientypen sind vorimplementiert, weitere können selbst hinzugefügt werden.

Architektur und Funktionsweise

1 Die Highlevel Server-Architektur von Apache Wink enthält eine Runtime-Schicht, die ankommende HTTP-Anfragen vom Host entgegen nimmt. Mit dieser Anfrage wird eine neue „Session“initiiert, indem ein Nachrichtenkontext erzeugt wird, der durch die Handler-Kette gereicht wird. Zunächst zu den Handlern, die dafür zuständig sind, Anfragen zu den richtigen Ressourcen und Methoden zuzuordnen. Wenn nötig werden eingehende Anfragen durch die entsprechenden Provider de-serialisiert. Nachdem die injizierbaren Parameter bereit sind, werden die zugeordneten Ressourcenmethoden aufgerufen und das generierte Antwort-Objekt wird wieder durch die Handler-Kette gereicht, bis es schließlich vom richtigen Provider als HTTP-Antwort serialisiert wird.

2 Apache Wink benötigt einen HTTP-Server, der gleichzeitig als Container für WebArchives dient (.war).

4.7.2. Entwicklung von REST basierten Anwendungen

Abschnitt	Stufe	Kommentar
Entwicklungsprozess/ Vorgehensmodell	0	
Modellierung von REST APIs	1	Laufzeitgenerierung von WADL zu einzelnen Ressourcen
Modellierungswerkzeuge	0	

Tabelle 4.33.: Auswertung Apache Wink Entwicklung von REST basierten Anwendunge

Entwicklungsprozess/Vorgehensmodell

1 Nein.

2 Nein.

3 Kann nicht beantwortet werden.

Modellierung von REST APIs

1 Wink unterstützt die Generierung von WADL.

2 Nein.

3 Kann nicht beantwortet werden.

4 Es können WADL-Dokumente für einzelne Ressourcen mittels HTTP OPTIONS Anfrage generiert werden, Desweiteren kann ein WADL-Dokument generiert werden, mittels *org.apache.wink.common.model.wadl.WADLGenerator*, das ein JAXB-Modell der Ressourcen-Klassen enthält, welches dann von Clients konsumiert werden kann.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Modellierungswerkzeuge

1 Nein.

2 Nein.

3 Nein.

4 Kann nicht beantwortet werden.

5 Nein.

6 Nein.

7 Nein.

8 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.7.3. Unterstützung grundlegender REST Prinzipien

Abschnitt	Stufe	Kommentar
Ressourcenidentifikation und Ressourcenstruktur	1	JAX-RS 1.1 Annotationen
Ressourcentypen	0	
Hypermedia	0	
Medientypen	1	JAX-RS 1.1 und weitere
Caching	1	Setzen von HTTP-CacheControl-Headern
Code-On-Demand	0	

Tabelle 4.34.: Auswertung Apache Wink Unterstützung grundlegender REST Prinzipien

Ressourcenidentifikation und Ressourcenstruktur

1 Gemäß JAX-RS erfolgt die Identifikation über URI-Elemente, die per @Path Annotation an Klassen und Methoden gebunden werden können. Dabei können sowohl einzelne URI-Elemente, als auch Teilpfade einer URI verwendet werden. Es ist damit möglich Aufruf-Hierarchien über verschiedene Klassen und Methoden hinweg zu realisieren.

2 Ja.

3 Siehe [jaxa, JAX-RS 1.1 Abschnitt 3.7].

4 Nein.

Ressourcentypen

1 Primärressourcen und Subressourcen können mit Mitteln aus JAX-RS 1.1 unkompliziert realisiert werden, sind aber nicht speziell als solche deklariert.

2 Gemäß JAX-RS 1.1 können Klassen mittels @Path Annotationen als Primärressource verfügbar gemacht werden.

3 Gemäß JAX-RS 1.1 können Methoden innerhalb von Klassen mittels @Path Annotationen als Subressourcenidentifikator genutzt werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Hypermedia

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Medientypen

1 Aktiv unterstützt werden *application/json*, *application/javascript*, *application/atomsvc+xml*, *application/atomcat+xml*, *application/atom+xml*, *application/xml*, *text/html*, *text/csv*, *application/opensearchdescription+xml*, *multipart/** und alle JAX-RS Medientypen.

2 Ja, es können eigene Provider (MessageBodyReader/-Writer) implementiert und registriert werden.

3 Wird gemäß der JAX-RS Spezifikation durchgeführt.

4 Wird gemäß der JAX-RS Spezifikation durchgeführt.

5 Wird gemäß der JAX-RS Spezifikation durchgeführt.

6 Wird gemäß der JAX-RS Spezifikation durchgeführt.

7 Wird gemäß der JAX-RS Spezifikation durchgeführt.

8 Wird gemäß der JAX-RS Spezifikation durchgeführt.

Caching

1 Unterstützt wird das Caching insofern, dass die Caching-Header von gesendeten Antworten entsprechend gesetzt werden könen.

2 Header müssen vom Entwickler selbst gesetzt werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Man kann die ETag-Header lesen und setzen.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Code-On-Demand

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.7.4. Erweiterte Technische Fähigkeiten

Abschnitt	Stufe	Kommentar
Protokollunterstützung jenseits von HTTP	0	
HTTP	2	JAX-RS 1.1
Unterstützung für Transaktionen	0	
Security	1	SSL, BasicAuth
Asynchronität	0	
Zuverlässigkeit	0	
Umgang mit großen Daten	0	

Tabelle 4.35.: Auswertung Apache Wink Erweiterte Technische Fähigkeiten

Protokollunterstützung jenseits von HTTP

1 Keine.

2 Kann nicht beantwortet werden..

3 Kann nicht beantwortet werden.

4 HTTP kann im Kontext einer REST-Applikation in vollem Umfang genutzt werden.

5 Kann nicht beantwortet werden.

HTTP

1 JAX-RS 1.1 definiert als Annotationen GET, PUT, POST, DELETE und HEAD. Desweiteren verlangt JAX-RS 1.1 auch noch Antworten auf OPTIONS Anfragen. Weiterhin können über die Annotation HttpMethod weitere HTTP-Methoden gemapped werden.

2 Ja.

1 Ja.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Unterstützung für Transaktionen

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Kann nicht beantwortet werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Security

1 Die Apache Wink Client API unterstützt SSL.

2 Der Client liefert einen BasicAuthSecurityHandler. Weitere Provider können selbst implementiert werden.

3 Kann nicht beantwortet werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Asynchronität

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Zuverlässigkeit

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Umgang mit großen Daten

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.8. Restlet

Eckdaten

URL	http://restlet.org
Lizenz	Apache 2.0, LGPL 3.0/2.1, EPL 1.0, CDDL 1.0
Entwickler	Restlet INC. (http://restlet.com/legal)

Tabelle 4.36.: Eckdaten Restlet

Beschreibung Restlet Framework unterstützt gleichzeitig die Implementierung von Servern und dazu passende Clients. Es gibt bei Restlet Framework bereits Klassen für Ressourcen, sowohl serverseitig als auch clientseitig. Serverseitig werden Anfragen über Annotationen an Instanzmethoden weitergeleitet.

Beispiel Ein kleines Beispiel ist in Listing 4.7.

Listing 4.7 Restlet Code Beispiel

```
public class Part03 extends ServerResource {

    public static void main(String[] args) throws Exception {
        // Create the HTTP server and listen on port 8182
        new Server(Protocol.HTTP, 8182, Part03.class).start();
    }

    @Get
    public String toString() {
        return "hello, world";
    }

}
```

4.8.1. Grundlagen

Abschnitt	Stufe	Kommentar
Allgemeines	2	Ausführliche Dokumentation, aktive Community auf StackOverflow, auf Erweiterungen ausgelegt
REST Server Applikationen	1	Standalone, Servlet, GWT (Google Web Toolkit), GAE (Google App Engine), Android und OSGi
REST Client Applikationen	1	Client-API mit Standardfunktionalität vorhanden, Server und Client können gemeinsame Interfaces verwenden

Tabelle 4.37.: Auswertung Restlet Grundlagen

Allgemeines

1 Ja, es gibt ein Tutorial, einen Benutzerleitfaden, eine Javadoc Beschreibung und ein Buch [LTB12].

2 Das Restlet Team bittet Nutzer, die Fragen bzgl. des Frameworks haben, Stackoverflow zu benutzen und dabei sollen sie den „restlet“tag anhängen. Darüber hinaus existiert eine Maillingliste auf tigris.org. [resc] Restlet wird gerade von SVN auf GIT umgestellt, dabei wird wiederum GitHub als Plattform und Anlaufstelle für Nutzer zur Verfügung gestellt.[resd]

3 Restlet wird unter mehreren Lizenzen angeboten. Diese sind Apache 2.0, LGPL 3.0, LGPL 2.1, CDDL 1.0, EPL 1.0. Das Restlet Team ermöglicht auch das Nutzen von kommerziellen Lizenzen, diese müssen dann aber mit den Entwicklern direkt ausgehandelt werden.

4 Restlet ist durch die darunter liegende Architektur stark Erweiterbar. Dabei besteht Restlet aus einer Restlet API und einer Restlet Engine auf dieser Restlet Extensions arbeiten. [LTB12, Seite 10 - 11]

REST Server Applikationen

1 Ja. Dies ist sehr einfach mit wenigen Zeilen Code mögliche.[rese]

2 Es gibt fertige Pakete für Java SE, Java EE Servlet Container, GWT (Google Web Toolkit), GAE (Google App Engine), Android und OSGI.

3 Portieren zwischen den Umgebungen Java SE, Java EE und GWT kann durch ändern des Initialisierungscodes bzw. des DeploymentDeskriptors ohne zusätzliche änderungen der Anwendung selbst erfolgen.

REST Client Applikationen

1 Ja das Framework beinhaltet eine ClientAPI. Sitzt man hinter einem Proxy oder ähnlichem ließe sich per Extension auch der ApacheHTTPClient nutzen.

2 Mit Restlet ist es möglich Interfaces zwischen Server und Client auszutauschen. Dabei muss das Interface mit den HTTP-Verben annotiert und von einer ServerResource implementiert werden. Der Client kann dieses Interface wrappen und eine Instanz der ServerResource erstellen.[resf]

3 String o.ä.wird automatisch konvertiert. Für speziellere Sachen wie JSON oder XML empfiehlt Restlet selbst die eigene XStream-Extension zu verwenden.

Architektur und Funktionsweise

1 Restlet besteht aus der Restlet API, Restlet Engine und aus einer Menge von Restlet Extensions. Restlet Applikationen sollen hauptsächlich auf der Restlet API aufbauen, können aber zusätzlich auf die Extensions zugreifen. ["rest in action"p. 10 - 11]. Restlet Applikationen sind Subklassen von ClientResource oder ServerResource und werden in Restlet in eine Komponente untergeordnet. Diese werden mittels Konnektoren auf URI's gelegt um diese ansprechbar zu machen. Bspw. Kann ein HTTP Konnektor eingehende Requests auf Methoden einer ServerResource weiterleiten, diese wiederum kann von ihr ausgehende Request an einen POP3 Konnektor weiterleiten. [LTB12, Seiten 8 - 17]

2 Der Restlet Kern baut rein auf Java SE auf. Zusätzlich existieren Editionen für andere Umgebungen, bspw. die OSGI Edition die den Kern als OSGI-Bundle anbieten.

4.8.2. Entwicklung von REST basierten Anwendungen

Abschnitt	Stufe	Kommentar
Entwicklungsprozess/ Vorgehensmodell	0	
Modellierung von REST APIs	1	Laufzeitgenerierung von WADL zu einzelnen Ressourcen
Modellierungswerkzeuge	0	

Tabelle 4.38.: Auswertung Restlet Entwicklung von REST basierten Anwendungen

Entwicklungsprozess/Vorgehensmodell

1 Nein.

2 In „Restlet in Action“Appendix D wird Resource-oriented-Analysis/Development (ROA/D) vorgestellt. Doch es wird nicht darauf eingegangen wie die identifizierten Artifakte des Vorgehensmodells auf Restlet abgebildet werden kann.

3 Nicht beantwortbar.

Modellierung von REST APIs

1 WADL wird mittels Extension unterstützt.[resg]

2 Nein. Es wird keine eigene IDL angeboten.

3 Um WADL zu erzeugen muss von den Klassen WADLServerResource, WADLApplication abgeleitet werden. Dann müssen noch Methoden implementiert werden, die die jeweiligen Methoden beschreiben. Per HTTP-OPTIONS kann dann die WADL abgerufen werden.

4 Es gibt keine Möglichkeit stubs aus WADL zu generieren mit Restlet.

Modellierungswerkzeuge

1 Nein.

2 Nein.

3 Nein.

4 Kann nicht beantwortet werden.

5 Nein.

6 Nein.

7 Nein.

8 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.8.3. Unterstützung grundlegender REST Prinzipien

Abschnitt	Stufe	Kommentar
Ressourcenidentifikation und Ressourcenstruktur	1	dedizierte Routing Objekte
Ressourcentypen	0	
Hypermedia	0	
Medientypen	2	Viele Medientypen
Caching	1	Setzen von HTTP-CacheControl-Headern
Code-On-Demand	0	

Tabelle 4.39.: Auswertung Restlet Unterstützung grundlegender REST Prinzipien

Ressourcenidentifikation und Ressourcenstruktur

1 Das Routing Konzept in Restlet, erlaubt es URIs sowohl in einer Root-Klasse zu implementieren als auch in einzelnen Ressourcen-Klassen, auf einkommende Request werden entsprechend des Mappings diese auf Methoden angewendet. URI-Templates werden von der Router-Klasse unterstützt.

2 Ja.

3 Um das Routing kümmern sich bei Restlet Objekte der Router-Klasse. In diesen Objekten können dann relative Pfade mittels .attach(String, Class) angehängt werden.Bei Objekten der Router-Klasse gibt es die Möglichkeit verschiedene „Matching-Modes “auszuwählen. Als default ist dabei *MODE_EQUALS* eingestellt, was bedeutet, dass die zu matchende URI exakt dem definierten Pattern entsprechen muss. Dies verbietet dann allerdings auch nachträgliches, bzw. darauf folgendes Routing. Um Hierarchisches Routing zu realisieren kann der Routing-Mode *MODE_STARTS_WITH* verwendet werden.

4 Nein.

Ressourcentypen

1 Nein. Es gibt nur die Klasse ServerResource, von der alle Ressourcen abgeleitet werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Hypermedia

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Es exisitiert eine Restlet Extension für RDF.

Medientypen

1 Restlet hat eine große Liste an bereits unterstützten Medientypen. [resh]

2 Eigene Medientypen können mittels Extensions und dem richtigen Konverter realisiert werden. Der Konverter muss dazu die entsprechende Serialisierung anbieten.

3 Setzen des Accept-Headers beim HTTP-Aufruf, ansonsten HTTP konform.

4 Man kann auf ServerResourcen „Varianten“setzen und diese dann gezielt bei Anfragen überprüfen und die geeignete Repräsentation erzeugen.

5 Von Hand.

6 HTTP konform.

7 HTTP konform.

8 HTTP konform.

Caching

1 Restlet erlaubt es Http-Caching zu nutzen, bietet aber derzeit noch keine eigenen Cache-Funktionalitäten. Diese sollen aber mit dem nächsten Release in Form der Klasse CacheService implementiert werden.

2 Ressourcenspezifisches Setzen der Wichtigsten Caching-Header (in den ServerResource-Klassen).

3 Keine. Nur Caching über HTTP.

4 Ja.

5 Nein.

Code-On-Demand

1 Es gibt keine Unterstützung für Code-On-Demand.

4.8.4. Erweiterte Technische Fähigkeiten

Abschnitt	Stufe	Kommentar
Protokollunterstützung jenseits von HTTP	0	
HTTP	2	HTTP im angemessenem Umfang nutzbar
Unterstützung für Transaktionen	0	
Security	1	SSL-Extension, BasicAuth und DigestAuth
Asynchronität	1	Non-Blocking-IO
Zuverlässigkeit	0	
Umgang mit großen Daten	2	

Tabelle 4.40.: Auswertung Restlet Erweiterte Technische Fähigkeiten

Protokollunterstützung jenseits von HTTP

1 Keine.

2 Nicht beantwortbar.

3 Nicht beantwortbar.

4 Nicht beantwortbar.

5 Nicht beantwortbar.

HTTP

1 Ja.

2 Ja.

1 Ja.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Unterstützung für Transaktionen

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Kann nicht beantwortet werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Security

1 Ja, es gibt eine SSL-Extension.

2 Basic, Digest.

3 Konfiguration durch Java oder Konfigurationsdatei.

4 Ja.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Asynchronität

1 Es gibt non-blocking NIO Modi. Asynchrone Anfragenbearbeitung.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Zuverlässigkeit

1 Gar nicht.

2 Nein.

3 Nein.

Umgang mit großen Daten

1 Ja, das Hochladen von Dateien wird in Restlet mittels der Apache FileUpload Extension ermöglicht. Die Extension benutzt dabei von Apache das Commons-Fileupload Packet [apab], das auch große Dateien beherscht.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.9. Play Framework

Eckdaten

URL	http://www.playframework.org
Lizenz	Apache 2.0 Lizenz
Entwickler	Zenexity (http://zenexity.fr/)

Tabelle 4.41.: Eckdaten Play

Listing 4.8 Play Beispiel Code

```
public class Application extends Controller {

  public static Result index() {
    return ok(index.render("Your new application is ready."));
  }

}
```

Beschreibung Das Framework versucht URI Pfade auf die Java Methoden abzubilden. Diese werden in einem Controller definiert und können so benutzt werden. Play baut auf HTTP auf und benutzt die gängigen Verben wie GET, PUT usw. Play geht auf die Entwicklung der Server ein. So bietet dieses Framework keine Möglichkeit einen Client zu bauen. Der Server beinhaltet einige Features sowie Contetn Negotiation und sogar Caching ist mit ein Paar Eingriffen sehr leicht umusetzen.

Beispiel Ein Auszug aus dem Tutorial von der Play Webseite:

In einer Console wird zum aufsetzen eines Servers müssen nur eingie Zeilen eingegeben werden.

- `play new myFirstApp`

  Erzeugt das Projekt mit allen nötigen Ressourcen und Unterordnern

- `cd myFirstApp`

  Betreten des Projekts

- `play`

  Führt die Console des Play Frameworks auf

Jetzt kann auf dem **http://localhost:9000/** der Server über einen Browser angesprochen werden. Die Routen/Pfade können in der conf/routes eingestellt werden. Ein Root Pfad in der Konfigurationsdatei würde so ausehen: **GET / controllers.Application.index()**. Nun braucht man einen Controller, welcher zum Root Pfad gemappt wurde, siehe Code Beispiel 4.8.

Weitere kleine Einstellungen und die Applikation ist bereit und läuft einwandfrei im Browser.

4.9.1. Grundlagen

Abschnitt	Stufe	Kommentar
Allgemeines	1	Ausführliche Dokumentation, aktive Community auf StackOverflow
REST Server Applikationen	1	Auf Standalone ausgelegt, Servlet möglich, aber nicht empfohlen
REST Client Applikationen	2	Client-API mit Standardfunktionalität, zusätzlich Async-HTTP-Client

Tabelle 4.42.: Auswertung Play Grundlagen

Algemeines

1 Auf der Playframework Webseite wird in einem 20-minütigen der Einstieg in die Web-Entwicklung mit dem Playframework geboten. Auf der Playframework Webseite ist zudem ausführliche Dokumentation vorhanden. Es gibt die beiden Bücher "Play for Java" und "Play for Scala".

2 Sehr aktive Community auf StackOverflow. http://www.playframework.com/get-involved Es gibt Mailing-Lists für User, Developer und für Security.

3 Apache 2.0 Lizenz.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

REST Server Applikationen

1 Ja. Das Playframework realisiert standardmäßig eine Standalone Anwendung. Verwendet wird dafür ein JBoss Netty HTTP Server. Mithilfe der Klasse ServletWrapper lässt sich Play allerdings auch in einen Servlet Kontext einfügen. Dies wird von Play! allerdings nicht empfohlen.

2 JVM, ansonsten mit Hilfe des (nicht empfohlenen) ServletWrappers in allen ServletContainern.

3 s.o.

REST Client Applikationen

1 Das Playframework liefert mit der Play WS API einen HTTP-Client. Mit dieser Api lässt sich auch der Async-HTTP-Client nutzen. [asy]

2 Highlevel-HTTP-Aufrufe, teilweise (De-)Serialisierung von Objekten.

3 Beispiel zur Nutzung der (De-)Serialisierung von Objekten.

```
HttpResponse res = WS.url("http://www.google.com").get();\\
String content = res.getString();\\
Document xml = res.getXml();\\
JsonElement json = res.getJson();\\
InputStream is = res.getStream();\\
```

Architektur und Funktionsweise

1 Die URIs werden in der conf/routes Datei festgelegt, die Java-Methoden, die die Aufrufe bearbeiten auch dort spezifiziert. Diese Methoden müssen dann in einer Controllerklasse implementiert werden. Da das Playframework als MVC Framework aufgebaut ist gibt es extra Packete für Datenmodell und Anzeige.

2 JBoss Netty Http Server, JVM, Darüber hinaus bietet Playframework viele andere Bibliotheken an: [plaa]

4.9.2. Entwicklung von REST basierten Anwendungen

Abschnitt	Stufe	Kommentar
Entwicklungsprozess/ Vorgehensmodell	0	
Modellierung von REST APIs	0	
Modellierungswerkzeuge	0	

Tabelle 4.43.: Auswertung Play Entwicklung von REST basierten Anwendungen

Entwicklungsprozess/Vorgehensmodell

1 Vom Playframework wird kein explizites Vorgehensmodell gefordert.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Modellierung von REST APIs

1 IDL's werden vom Playframework nicht nativ unterstützt.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Modellierungswerkzeuge

1 Nein.

2 Nein.

3 Nein.

4 Kann nicht beantwortet werden.

5 Nein.

6 Nein.

7 Nein.

8 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4.9.3. Unterstützung grundlegender REST Prinzipien

Abschnitt	Stufe	Kommentar
Ressourcenidentifikation und Ressourcenstruktur	1	Zentrale Routingdatei
Ressourcentypen	0	
Hypermedia	0	
Medientypen	1	Standard Medientypen
Caching	1	Eigene Caching-API
Code-On-Demand	0	

Tabelle 4.44.: Auswertung Play Unterstützung grundlegender REST Prinzipien

Ressourcenidentifikation und Ressourcenstruktur

1 Alle URIs werden in der conf/routes Datei verwaltet. Dabei muss das HTTP-Verb, die relative URI ab Server und die Methode, welche den Aufruf bearbeitet (fqn), angegeben werden.

2 Es ist im Playframework möglich bei der URI-Gestaltung dynamische Teile und sogar reguläre Ausdrücke zu verwenden. Ebenfalls ist es möglich optionale URI-Teile zu nutzen.

3 Alle URI sind in der conf/routes Datei gespeichert. Ebenso die zugehörigen Java-Methoden. Soll innerhalb des Frameworks ein request umgeleitet werden, so kann beispielsweise die URI einer Methode mit fqn controllers.admin.Application.hello zur Laufzeit durch controllers.admin.routes.Application.hello() bestimmt werden.

4 Nein.

Ressourcentypen

1 Es gibt nur die Klasse Controller, in der die Java-Methoden zur Anfragen-Bearbeitung implementiert werden können.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Hypermedia

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Medientypen

1 Play kann folgende Medientypen von Haus aus *text/html*, *application/xhtml*, *text/xml*, *application/xml*, *text/plain*, *text/javascript*, *application/json* verarbeiten, damit Entwickler diese komfortabel nutzen können.

2 Ja, aber es muss alles “von Hand” gemacht werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Playframework empfiehlt Content-Negotiation über die URL zu regeln. Dazu sollen dann URI-Templates der Form: path/file.{format} verwendet werden. Es ist aber auch möglich HTTP-Content-Negotiation zu nutzen. Dann werden Accept-Header beachtet.[plab] [plac] [plad]

5 HTML wird, sofern vorhanden (auch bei einer wildcard) per default ausgewählt, sonst werden die in der *Routes* Datei gesetzten Mappigs genutzt.

6 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

7 Kann beim Setzen des Content-Types der Antwort mitgesetzt werden. UTF-8 wird per default gesetzt.

8 Accept-Language Header. Best-Match (falls verfügbar, ansonsten default.)

Caching

1 Das Playframework bietet eine eigene Cashing-Api die von EHCache [EhCb] implementiert wird wird. Die API erlaubt Objekte in den cache zu schreiben (auch temporär), lesen und löschen. Ebenso kann dieser Cache auf bestimmte Nutzer (von außen) eingeschränkt werden.

2 Caching-Header können auch in der conf/application.conf gesetzt werden.

3 Play benutzt EHCache zum cachen. EHCache untersützt alle Caching-Header aus HTTP 1.1, somit werden Validierungs- und Expirationsmodell unterstützt.

4 Ja.

5 In der conf/application.conf kann die automatische Generierung von E-Tags aktiviert werden.

Code-On-Demand

1 Das Playframework bietet keine spezielle Unterstützung für Code-On-Demand Lösungen.

4.9.4. Erweiterte Technische Fähigkeiten

Abschnitt	Stufe	Kommentar
Protokollunterstützung jenseits von HTTP	0	
HTTP	2	HTTP im angemessenem Umfang nutzbar
Unterstützung für Transaktionen	1	Automatische JPA Nutzung bei Anfragen an den Server
Security	1	SSL, Zertifikate, Authentifizierungen
Asynchronität	2	Non-Blocking-IO, Asynchroner Nachrichtenaustausch mit Akka
Zuverlässigkeit	0	
Umgang mit großen Daten	2	Auslagerung auf Festplatte automatisch

Tabelle 4.45.: Auswertung Play Erweiterte Technische Fähigkeiten

Protokollunterstützung jenseits von HTTP

1 Keine.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

4 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

HTTP

1 Laut Dokumentation auf der Playframework-Webseite werden Get, Put, Post, Delete und Head unterstützt. Durch manuelles Bearbeiten des Parsers der routes-File lassen sich allerdings auch andere Verben nutzen.

2 Da HTTP-Aufrufe nur auf Java-Methoden ümgeleitet"werden, bleibt die Einhaltung der Verbsemantik in Bezug auf Idempotenz und Sicherheit in der Verantwortung des Entwicklers.

1 Ja.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Unterstützung für Transaktionen

1 Play bietet JPA persistence und Ebean, mit dem transaktionale Datenhaltung ermöglicht werden kann. Bei der Nutzung von Ebean muss das transaktionale Verhalten allerdings selbst bestimmt werden. (per @play.db.ebean.Transactional Annotation an Methoden) oder "von Hand" mit:

```
Ebean.execute(new TxRunnable() { }});
```

[Plae] [plaf]

2 Das Playframework startet automatisch eine JPA-Transaction bei einer Anfrage und führt den Commit aus, wenn der Java-Code keine Exception geworfen hat und die Antwort versandt wurde.

3 Nein.

4 Nein.

Security

1 Das Framework unterstützt SSL und Zertifikat-Überprüfung. Dazu müssen Die Zertifikat mit einem bestimmten Namen im conf/ verzeichnis der Anwendung gespeichert werden. Ebenso muss in der conf/Application.conf ein HTTPS Port eingetragen werden.[plag]

2 Der Webserver kann auch in der conf/Application.conf konfiguriert werden.[plah]

3 Playframework bietet dafür extra ein Sicherheitsmodul, mit dem sich Authentifizierung beim Server (auch mit verschiedenen Rollen) realisieren lässt. [plai]

4 In der Konfigurationsdatei kann mittels application.secure="value" ein geheimer Schlüssel konfiguriert werden. Ist dieser nicht gesetzt, wird play.libs.Crypto.sign keine Nachrichten verschlüsseln. Ob automatisch Nachrichten verschlüsselt werden ist nicht klar erkennbar.[plaj]

5 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Asynchronität

1 Ja, Playframework unterstützt unter anderem Non-Blocking I/O.[plak][plal]

2 Mit Hilfe von Akka(in Play integriert) lässt sich auch asynchroner Nachrichtenaustausch realisieren.

Zuverlässigkeit

1 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

3 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

Umgang mit großen Daten

1 Ja, das Playframework bietet Möglichkeiten Dateien "chunked" zu übertragen. Zusätzlich werden große Dateien auf der Festplatte ausgelagert.[plam] [plan] [plao]

2 Konnte im Rahmen der Fachstudie nicht evaluiert werden.

5. Ergebnisse

5.1. Kommentar zur Bearbeitung

Zur Beantwortung der Fragen aus dem Kriterienkatalog haben wir fast ausschließlich Informationen herangezogen, die zum Zeitpunkt der Fachstudie online verfügbar waren. Dazu gehören die Webseiten der einzelnen Frameworks, aber auch Community-Seiten, wie bspw. [stackoverflow], Blogs, Mailinglist-Archive, [googlegroups] etc.
Aufgrund der Tatsache, dass wir nur frei verfügbare Informationen genutzt haben, war es teilweise sehr schwer, Informationen zu speziellen Themen zu finden. Deshalb konnten wir nicht alle Fragen definitiv beantworten.
Die Kategorie [Architektur und Funktionsweise] haben wir nicht zur Auswertung herangezogen, weil jeder Entwickler und jedes Projekt andere Anforderungen an ein Framework hat.

5.2. Grundlagen

5.2.1. Allgemeines - Ergebnisse

Die Frameworks, die bei der Evaluierung am besten abgeschnitten haben sind Jersey, CXF, RESTEasy, Restlet und VRaptor. Alle der genannten Frameworks zeichneten sich durch eine ausführliche Dokumentation aus, mit deren Hilfe man erste Services entwickeln kann. Desweiteren war im Netz auch genügend Informationen vorhanden, um auch größere Projekte realisieren zu können. Jersey, CXF und RESTEasy hatten dabei den Vorteil, dass durch die Implementierung von JAX-RS 1.1 sehr viele Hilfestellungen verfügbar waren. Frameworks, die JAX-RS 1.1 nicht implementierten waren dennoch ausreichend dokumentiert um Services zu entwickeln, seien es einfache oder schwere Projekte. Die Communities der Frameworks waren subjektiv bei den JAX-RS Frameworks größer, da prinzipiell Lösungen von einem Framework auf andere übertragen werden können. VRaptor und Restlet hingegen, zeichneten sich durch kleinere, aber konzentriertere Gruppen, deren Aktivität subjektiv höher war. Alle Frameworks standen unter Lizenzen, die für die meisten Projekte nutzbar wären. Bei der prinzipiellen Erweiterbarkeit der Frameworks, waren große Unterschiede bemerkbar. Einige Frameworks wie Restlet sind komplett darauf ausgelegt durch Extensions erweitert zu werden, bei anderen konnten Informationen diesbezüglich nicht ohne eine intensive Analyse des Codes evaluiert werden.

5.2.2. REST Server Applikationen - Ergebnisse

Hervorzuhebende Frameworks in dieser Kategorie waren CXF und Restlet. Mit beiden Frameworks ist es möglich, entwickelte Applikationen Standalone, als Servlet, oder innerhalb eines OSGi Frameworks zu nutzen. Restlet bietet darüberhinaus noch weitere Möglichkeiten Applikationen zu betreiben, wie bspw. in der Google App Engine. Die meisten anderen Frameworks waren meist auf Servlet-Betrieb beschränkt und müssten über speziellere Konfigurationen für den Standalone-Betrieb eingestellt werden. Andere Umgebungen werden nur von sehr wenigen Frameworks angeboten.

5.2.3. REST Client Applikationen - Ergebnisse

Alle Frameworks außer Scooter verfügen über einen dedizierten Client, der mindestens die Funktionalität eines Standard-HTTP-Clients anbietet. Hervorzuheben sind Jersey, VRaptor, CXF und Play, die speziellere APIs für Client die Cliententwicklung anbieten. Im Umfang dieser APIs waren meist Zusatzfunktionen enthalten, die sich mit der Serialisierung von Objekten beschäftigten. Einige unterstützen den Austausch gemeinsamer Interfaces zwischen Server und Client. Der Hypermedia-Aspekt von REST war nur im Restfulie Clientframework vertreten, das als Client für VRaptor genutzt wird.

5.3. Entwicklung von REST basierten Anwendungen

5.3.1. Entwicklungsprozess/ Vorgehensmodell - Ergebnisse

Bei keinem Framework gab es Vorgaben oder Empfehlungen bestimmte Entwicklungsprozesse zu nutzen.

5.3.2. Modellierung von REST APIs - Ergebnisse

Nur Jersey, CXF, Wink und Restlet bieten Unterstützung für WADL. Abgesehen von CXF beschränkt sich diese Unterstützung auf die Laufzeitgenerierung von WADL-Dokumenten. Weiterhin gibt es das Tool WADL2Java, das in der Lage ist aus WADL-Beschreibungen Stubs zu generieren. Dieses Tool kann bspw. für CXF und Jersey genutzt werden.

5.3.3. Modellierungswerkzeuge - Ergebnisse

Keines der untersuchten Frameworks beinhaltet Modellierungswerkzeuge.

5.4. Unterstützung grundlegender REST Prinzipien

5.4.1. Ressourcenidentifikation und Ressourcenstruktur - Ergebnisse

Bei allen Frameworks funktioniert das Routing von Anfragen ähnlich, dabei unterschieden sich diese sich nur in der Deklaration. Einige benutzten Java-Annotationen (Alle JAX-RS Frameworks, Restlet, VRaptor), andere eine zentrale Routing/Konfigurations-Datei. Dabei werden ganze oder Fragmente von Pfaden auf Java-Methoden gemappt. Einzig das Scooter Framework mit seinem Conventions-over-Configuration Ansatz generierte Routings automatisch auf Modellklassen ohne das Routing von Hand zu deklarieren. Weiterhin unterstützten alle Frameworks die Verwendung von URI Templates. Andere Arten der Identifizierung von Ressourcen, außer URI, unterstützte keines der Frameworks.

5.4.2. Ressourcentypen - Ergebnisse

Prinzipiell konnten alle Frameworks Primär- und Subressourcen deklarieren. Die erweiterten Konzepte für Ressourcen aus [Tilkov], wurden von keinem Framework außer Scooter unterstützt, wobei auch Scooter nicht alle Ressourcentypen unterstützt. Bei Scooter ist es bspw. möglich Listenressourcen in der Konfigurationsdatei eines Projekts zu definieren.

5.4.3. Hypermedia - Ergebnisse

In der Hypermedia Kategorie ist RESTEasy hervorzuheben. RESTEasy ermöglicht es, mittels Annotationen, Links (Atom Links, Link Header) in Repräsentationen einzufügen. Dabei konnten diese auch zustandsabhängig eingefügt und mit eigenen Linkrelationen typisiert werden. Jersey bietet das Einfügen von Link-Headern mittels Annotation. Bei den anderen Frameworks muss der Hypermedia-Aspekt gänzlich selbst entwickelt werden.

5.4.4. Medientypen - Ergebnisse

Alle Frameworks unterstützen das Verarbeiten von XML und JSON Medientypen und ermöglichen es, falls nötig, eigene Medientypen zu definieren. Restlet bietet zusätzlich noch eine Reihe von Medientypen, die in anderen Frameworks nicht standardmäßig enthalten sind.

5.4.5. Caching - Ergebnisse

Alle Frameworks verfügen über eine Abstraktion von HTTP-CachControl Headern, nur wenige automatisierten Caching komplett. Hervorzuheben sind sowohl REASTEasy und Resthub, mit deren Hilfe Ressourcen per Annotation für das Caching konfiguriert werden

können, als auch Scooter, wo Caching-Optionen in der Routes-Datei verfügbar sind. Weiterhin können bspw. Caching-Frameworks, wie EHCache, in CXF integriert werden.

5.4.6. Code-On-Demand - Ergebnisse

Der optionale Code-On-Demand REST Aspekt wird von keinem Framework direkt unterstützt.

5.5. Erweiterte Technische Fähigkeiten

5.5.1. Protokollunterstützung jenseits von HTTP - Ergebnisse

Kein Framework, ausser CXF, ermöglicht das Nutzen von anderen Protokollen ausser HTTP. CXF kann als Protokoll JMS verwenden und gibt auch Informationen wie man Queues und Topics dafür konfiguriert

5.5.2. HTTP - Ergebnisse

Alle Frameworks ermöglichen das Nutzen von HTTP in vollem Umfang.

5.5.3. Unterstützung für Transaktionen - Ergebnisse

Die Frameworks, die auf Spring aufbauen, VRaptor und Resthub, bieten vorgefertigte Lösungen für die Nutzung von Transaktionen. Play startet für jede Anfrage an den Server automatisch eine Transaktion. Bei den anderen Frameworks gab es zum Teil gut dokumentierte Lösungswege, Transaktionen zu integrieren. Eine eigene Transaktions-API oder eigens definierte Annotationen werden aber von keinem der evaluierten Frameworks zur Verfügung gestellt.

5.5.4. Security - Ergebnisse

Alle evaluierten Frameworks sind in der Lage SSL zu nutzen. Verwendet werden dazu meist Funktionen aus dem Servlet-Kontext. Ebenso bieten die meisten Frameworks zur Authentifizierung zumindest BasicAuth an. Hervorzuheben sind RESTEasy und CXF. RESTEasy stellt Verschlüsselungs- und Signaturfunktionalität bereit, nei CXF gibt es Funktionalität zur Verschlüsselung und Signatur von XML.

5.5.5. Asynchronität - Ergebnisse

Asynchrone Request Verarbeitung wurde von den meisten Frameworks unterstützt. Nur RESTEasy, Resthub und Play bieten vorgefertigte Möglichkeiten asynchronen Nachrichtenaustausch zu realisieren.

5.5.6. Zuverlässigkeit - Ergebnisse

Für die Handhabung der, in [Tilkov] beschriebenen, Zuverlässigkeitsprobleme, bietet keines der evaluierten Frameworks aktiv Unterstützung.

5.5.7. Umgang mit großen Daten - Ergebnisse

Außer Resthub, Wink und Jersey, ist es bei allen Frameworks möglich, die Festplatte zur Verarbeitung großer Dateien zu nutzen. Jersey nutzt zur Handhabung großer Dateien eine eigene Annotation, Resthub arbeitet mit Streams.

6. Zusammenfassung und Ausblick

Diese Studie hat einen Überblick über die verschiedenen REST Frameworks im JAVA Umfeld gegeben. Mit Hilfe der Kriterien sollte es dem Leser einfacher fallen, für seine REST Applikation, das passende Framework zu finden. Die Beantwortung der Fragen und die Bewertung mit den Noten 0, 1 und 2 besagen wie stark, das eine oder andere Thema unterstütz wird. Die Frage „Welches Framework ist das bessere?“können wir hier nicht beantworten. Es gibt viele unterschiedliche Frameworks, welche ihre eigenen Sträken und Schwächen haben. Dabei hängt es vom Projekt ab, in dem ein Framework eingesetzt werden soll, welches Framework mit seinen Stärken am besten passt. Einige Schwächen fallen jedoch bei allen Frameworks auf, die fehlenden Modellierungswerkzeuge so wie Vorgehensmodelle um eine REST Applikation implementieren zu können. In [ZBD11] stellt Zuzak ein Vorgehensmodell, dieses muss jedoch ohne spezielle Tools oder IDEs vollzogen werden, so auch die Modellierung der Ressourcen oder des URI Designs. Hier könnte sich die REST Community zusammenschließen und solche Werkzeuge anbieten, damit die Applikation, die entsteht, dann wirklich auch RESTful wird und nicht nur eine HTTP basierte Schnittstelle repräsentiert.[zus]

Ausblick

Die vorgestellten Frameworks werden als aktiv bezeichnet, so kann man hier gespannt sein, wie diese sich weiterentwickeln. JAX-RS 2.0 wird wohl von allen JAX-RS-Frameworks implementiert werden. In dieser Version ist bspw. ein einfacherer und verbesserter Client spezifiziert [jaxb]. Auch die anderen Frameworks entwickeln sich weiter, denn nicht umsonst befinden sich diese in den Versionen > 1.0. Vielleicht wird es eines Tages das Ultimative Framework geben, das die Stärken aller Frameworks vereint, aber bis dahin haben die Entwickler von REST Applikationen die Quahl der Wahl.

A. Gesamtübersicht aller Auswertungen

Framework	Stufe	Kommentar
Jersey	2	Ausführliche Dokumentation, aktive Community, Referenzimplementierung von JAX-RS, gut erweiterbar
Scooter	1	Gute Dokumentation, semi-Aktive Community, erweiterbar
VRaptor	2	Gute Dokumentation und Tutorials, erweiterbar, aktive Community
Resthub	1	Gute Dokumentation
CXF	2	Ausführliche Dokumentation, aktive Community, explizit auf Erweiterungen ausgelegt
RESTEasy	2	Ausführliche Dokumentation, aktive Community
Wink	1	Dokumentation vorhanden, auf Erweiterung ausgelegt
Restlet	2	Ausführliche Dokumentation, aktive Community auf StackOverflow, auf Erweiterungen ausgelegt
Play	1	Ausführliche Dokumentation, aktive Community auf StackOverflow

Tabelle A.1.: Auswertungen in der Kategorie Allgemeines

Framework	Stufe	Kommentar
Jersey	1	Auf Servlet ausgelegt, Standalone möglich
Scooter	1	Auf Standalone ausgelegt, Servlet aufwändig
VRaptor	2	Auf Servlet ausgelegt, Standalone sehr aufwändig
Resthub	1	Nur Servlet möglich
CXF	2	Servlet und Standalone komfortabel möglich, OSGi möglich
RESTEasy	1	Servlet Container
Wink	1	Auf Servlet ausgelegt
Restlet	2	Standalone, Servlet, GWT (Google Web Toolkit), GAE (Google App Engine), Android und OSGi
Play	1	Auf Standalone ausgelegt, Servlet möglich, aber nicht empfohlen

Tabelle A.2.: Auswertungen in der Kategorie REST Server Applikationen

Framework	Stufe	Kommentar
Jersey	1	Client-API, Komfortfunktionen, gute Erweiterungsmöglichkeiten
Scooter	1	keine dedizierte Client-API, auf Browser-Clients ausgelegt
VRaptor	2	Eigenes Clientprojekt Restfulie
Resthub	1	Auf Browser-Clients ausgelegt. Eine große Bibliothek Backbone.js wird bereitgestellt.
CXF	2	Verschiedene Clients mit unterschiedlichen Funktionalitäten für verschiedene Ansprüche
RESTEasy	1	Client Framework wird angeboten, Server und Client können gemeinsame Interfaces verwenden
Wink	1	Client-API mit Standardfunktionalität vorhanden
Restlet	2	Client-API mit Standardfunktionalität vorhanden, Server und Client können gemeinsame Interfaces verwenden
Play	1	Client-API mit Standardfunktionalität, zusätzlich Async-HTTP-Client

Tabelle A.3.: Auswertungen in der Kategorie Client Applikationen

Framework	Stufe	Kommentar
Jersey	0	-
Scooter	0	-
VRaptor	0	-
Resthub	0	-
CXF	0	-
RESTEasy	0	-
Wink	0	-
Restlet	0	-
Play	0	-

Tabelle A.4.: Auswertungen in der Kategorie Entwicklungsprozess/Vorgehensmodell

Framework	Stufe	Kommentar
Jersey	1	Laufzeitgenerierung von WADL zu einzelnen Ressourcen
Scooter	0	-
VRaptor	0	-
Resthub	0	-
CXF	2	erlaubt von Code zu WADL und WADL zu Code Generierung
RESTEasy	0	-
Wink	1	Laufzeitgenerierung von WADL zu einzelnen Ressourcen
Restlet	1	Laufzeitgenerierung von WADL zu einzelnen Ressourcen
Play	0	-

Tabelle A.5.: Auswertungen in der Kategorie Modellierung von REST API's

Framework	Stufe	Kommentar
Jersey	0	-
Scooter	0	-
VRaptor	0	-
Resthub	0	-
CXF	0	-
RESTEasy	0	-
Wink	0	-
Restlet	0	-
Play	0	-

Tabelle A.6.: Auswertungen in der Kategorie Modellierungswerkzeuge

Framework	Stufe	Kommentar
Jersey	1	JAX-RS 1.1 Annotationen
Scooter	2	Conventions-over-Configuration, Klassen bekommen ihr URI anhand der Struktur im Projekt. URI-Konfiguration auch explizit möglich
VRaptor	1	Annotationen ähnlich zu JAX-RS 1.1
Resthub	1	Zentrale Routingdatei
CXF	1	JAX-RS 1.1 Annotationen
RESTEasy	1	JAX-RS 1.1 Annotationen
Wink	1	JAX-RS 1.1 Annotationen
Restlet	1	dedizierte Routing Objekte,Annotatioen
Play	1	Zentrale Routingdatei

Tabelle A.7.: Auswertungen in der Kategorie Ressourcenidentifikation und Ressourcenstruktur

Framework	Stufe	Kommentar
Jersey	0	-
Scooter	2	Conventions-over-Configuration Ansatz ermöglicht viele Ressourcentypen
VRaptor	0	-
Resthub	0	-
CXF	0	-
RESTEasy	0	-
Wink	0	-
Restlet	0	-
Play	0	-

Tabelle A.8.: Auswertungen in der Kategorie Ressourcentypen

Framework	Stufe	Kommentar
Jersey	1	Setzen von Link-Headern mittels Annotation möglich
Scooter	0	-
VRaptor	1	Spezielle Hypermedia-Ressource
Resthub	0	-
CXF	0	-
RESTEasy	2	Atom Links, LinkHeader, Zustandsabhängig Links
Wink	0	-
Restlet	0	-
Play	0	-

Tabelle A.9.: Auswertungen in der Kategorie Hypermedia

Framework	Stufe	Kommentar
Jersey	1	JAX-RS 1.1
Scooter	1	Standard Medientypen
VRaptor	1	Standard Medientypen, benutzt zentrale Medientypen Repository
Resthub	1	Standard Medientypen
CXF	1	JAX-RS 1.1
RESTEasy	1	JAX-RS 1.1 und weitere
Wink	1	JAX-RS 1.1 und weitere
Restlet	2	Viele Medientypen
Play	1	Standard Medientypen

Tabelle A.10.: Auswertungen in der Kategorie Medientypen

Framework	Stufe	Kommentar
Jersey	1	CacheControl Klasse als Abstraktion von Caching-Headern
Scooter	2	Per Routes Datei konfigurierbar
VRaptor	1	ObservableResource Klasse, um Caching zu behandeln.
Resthub	2	Per Annotation konfigurierbar
CXF	1	Caching durch Cachingframework
RESTEasy	2	Per Annotation konfigurierbar
Wink	1	Setzen von HTTP-CacheControl-Headern
Restlet	1	Setzen von HTTP-CacheControl-Headern
Play	1	Eigene Caching API

Tabelle A.11.: Auswertungen in der Kategorie Caching

Framework	Stufe	Kommentar
Jersey	0	-
Scooter	0	-
VRaptor	0	-
Resthub	0	-
CXF	0	-
RESTEasy	0	-
Wink	0	-
Restlet	0	-
Play	0	-

Tabelle A.12.: Auswertungen in der Kategorie Code-On-Demand

Framework	Stufe	Kommentar
Jersey	0	-
Scooter	0	-
VRaptor	0	-
Resthub	0	-
CXF	2	JMS
RESTEasy	0	-
Wink	0	-
Restlet	0	-
Play	0	-

Tabelle A.13.: Auswertungen in der Kategorie Protokollunterstützung jenseits von HTTP

Framework	Stufe	Kommentar
Jersey	2	JAX-RS 1.1
Scooter	2	HTTP im angemessenem Umfang nutzbar
VRaptor	2	HTTP im angemessenem Umfang nutzbar
Resthub	2	HTTP im angemessenem Umfang nutzbar
CXF	2	JAX-RS 1.1
RESTEasy	2	JAX-RS 1.1
Wink	2	JAX-RS 1.1
Restlet	2	HTTP im angemessenem Umfang nutzbar
Play	2	HTTP im angemessenem Umfang nutzbar

Tabelle A.14.: Auswertungen in der Kategorie HTTP

Framework	Stufe	Kommentar
Jersey	0	-
Scooter	1	JTA, JDBC
VRaptor	1	Spring, JPA
Resthub	2	Spring
CXF	0	-
RESTEasy	0	-
Wink	0	-
Restlet	0	-
Play	1	Automatische JPA Nutzung bei Anfragen an den Server

Tabelle A.15.: Auswertungen in der Kategorie Unterstützung von Transaktionen

Framework	Stufe	Kommentar
Jersey	1	SSL
Scooter	1	Authentifikation
VRaptor	2	SSL und Plugins für Authentifizierung
Resthub	1	SSL
CXF	2	SSL,Keystore, Signaturen,viele Authentifizierungen XML-Verschlüsselung und XML-Signaturen
RESTEasy	2	SSL, OAuth, Signaturen, Verschlüsselung
Wink	1	SSL, BasicAuth
Restlet	1	SSL-Extension, BasicAuth und DigestAuth
Play	1	SSL, Zertifikate, Authentifizierungen

Tabelle A.16.: Auswertungen in der Kategorie Security

Framework	Stufe	Kommentar
Jersey	1	Non-Blocking Client
Scooter	0	-
VRaptor	1	Non-Blocking Requestverarbeitung
Resthub	2	Non-Blocking
CXF	1	Unterstützung für Apache HttpAsyncClient (kann Non-Blocking I/O)
RESTEasy	2	Non-Blocking-IO, Asynchrone Requests
Wink	0	-
Restlet	1	Non-Blocking-IO
Play	2	Non-Blocking-IO, Asynchroner Nachrichtenaustausch mit Akka

Tabelle A.17.: Auswertungen in der Kategorie Asynchronität

Framework	Stufe	Kommentar
Jersey	0	-
Scooter	0	-
VRaptor	0	-
Resthub	0	-
CXF	0	-
RESTEasy	0	-
Wink	0	-
Restlet	0	-
Play	0	-

Tabelle A.18.: Auswertungen in der Kategorie Zuverlässigkeit

Framework	Stufe	Kommentar
Jersey	2	Extra Annotation
Scooter	2	Auslagerung auf Festplatte möglich
VRaptor	2	Auslagerung auf Festplatte möglich
Resthub	1	Per Stream
CXF	2	Auslagerung auf Festplatte möglich
RESTEasy	2	Auslagerung auf Festplatte automatisch
Wink	0	-
Restlet	2	Auslagerung auf Festplatte automatisch
Play	2	Auslagerung auf Festplatte automatisch

Tabelle A.19.: Auswertungen in der Kategorie Umgang mit großen Daten

Literaturverzeichnis

[apaa] http://hc.apache.org/httpcomponents-asyncclient-dev/index.html. (Zitiert auf Seite 76)

[apab] http://commons.apache.org/proper/commons-fileupload/. (Zitiert auf Seite 103)

[asy] https://github.com/AsyncHttpClient/async-http-client. (Zitiert auf Seite 106)

[cxfa] http://cxf.apache.org/. (Zitiert auf Seite 67)

[cxfb] http://cxf.apache.org/docs/jax-rs.html. (Zitiert auf Seite 68)

[cxfc] http://cxf.apache.org/docs/rest-with-jax-ws-provider-and-dispatch.html. (Zitiert auf Seite 68)

[cxfd] http://cxf.apache.org/docs/http-binding.html. (Zitiert auf Seite 68)

[cxfe] http://cxf.apache.org/docs/restful-services.html. (Zitiert auf Seite 68)

[cxff] http://cxf.apache.org/mailing-lists.html. (Zitiert auf Seite 68)

[cxfg] http://cxf.apache.org/docs/cxf-architecture.html#CXFArchitecture-Bus. (Zitiert auf den Seiten 68 und 69)

[cxfh] http://svn.apache.org/repos/asf/cxf/trunk/distribution/src/main/release/samples/java_first_jaxws/src/main/java/demo/hw/server/Server.java. (Zitiert auf Seite 68)

[cxfi] http://cxf.apache.org/docs/application-server-specific-configuration-guide.html. (Zitiert auf Seite 68)

[cxfj] http://cxf.apache.org/docs/cxf-architecture.html#CXFArchitecture-Frontends. (Zitiert auf Seite 69)

[cxfk] http://cxf.apache.org/docs/cxf-architecture.html#CXFArchitecture-Messaging%26Interceptors. (Zitiert auf Seite 69)

[cxfl] http://cxf.apache.org/docs/cxf-architecture.html#CXFArchitecture-TheServiceModel. (Zitiert auf Seite 69)

[cxfm] http://cxf.apache.org/docs/data-binding-architecture.html. (Zitiert auf Seite 69)

[cxfn] http://cxf.apache.org/docs/cxf-architecture.html#CXFArchitecture-DataBindings. (Zitiert auf Seite 69)

[cxfo] http://cxf.apache.org/docs/cxf-architecture.html\T1\ss#CXFArchitecture-ProtocolBindings. (Zitiert auf Seite 69)

[cxfp] http://cxf.apache.org/docs/cxf-architecture.html#CXFArchitecture-Transports. (Zitiert auf Seite 69)

[cxfq] http://cxf.apache.org/docs/jaxrs-services-description.html. (Zitiert auf Seite 70)

[cxfr] http://cxf.apache.org/docs/jax-rs-basics.html#JAX-RSBasics-Customselectionbetweenmultipleresources. (Zitiert auf Seite 71)

[cxfs] http://cxf.apache.org/docs/jax-rs-advanced-features.html#JAX-RSAdvancedFeatures-JMSSupport. (Zitiert auf Seite 74)

[cxft] http://cxf.apache.org/docs/jax-rs-advanced-features.html#JAX-RSAdvancedFeatures-JMSSupport. (Zitiert auf Seite 74)

[cxfu] http://cxf.apache.org/docs/secure-jax-rs-services.html#SecureJAX-RSServices-ValidatingBasicAuthcredentialswithSTS. (Zitiert auf Seite 75)

[cxfv] http://cxf.apache.org/docs/jax-rs-failover.html. (Zitiert auf Seite 76)

[cxfw] http://cxf.apache.org/docs/jax-rs-multiparts.html#JAX-RSMultiparts-Readinglargeattachments. (Zitiert auf Seite 76)

[ehca] http://www.ehcache.org/documentation/user-guide/web-caching. (Zitiert auf Seite 73)

[EhCb] http://ehcache.org/. (Zitiert auf Seite 110)

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1, 1999. URL http://www.rfc.net/rfc2616.html. (Zitiert auf Seite 24)

[Fieoo] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. Dissertation, 2000. AAI9980887. (Zitiert auf den Seiten 11 und 24)

[ian] www.iana.org/assignments/media-types. (Zitiert auf Seite 15)

[imp] http://code.google.com/p/implementing-rest/wiki/ByLanguage. (Zitiert auf Seite 11)

[jaxa] https://jax-rs-spec.java.net/. (Zitiert auf den Seiten 29, 34, 36, 71, 80 und 89)

[jaxb] https://jax-rs-spec.java.net/nonav/2.0/apidocs/index.html. (Zitiert auf Seite 121)

[jera] http://jersey.java.net/nonav/documentation/latest/user-guide.html#client-api. (Zitiert auf Seite 31)

[jerb] https://wikis.oracle.com/display/Jersey/WADL. (Zitiert auf Seite 33)

[jerc] http://wadl.java.net/wadl2java.html. (Zitiert auf Seite 33)

[jerd] http://docs.oracle.com/cd/E19776-01/820-4867/ggqnw/index.html. (Zitiert auf Seite 34)

[jere] http://docs.oracle.com/cd/E19776-01/820-4867/ggqny/index.html. (Zitiert auf Seite 34)

[jerf] http://jersey.java.net/nonav/documentation/latest/user-guide.html#d4e1482. (Zitiert auf Seite 35)

[jerg] http://jersey.java.net/nonav/apidocs/1.5/jersey/javax/ws/rs/core/MediaType.html. (Zitiert auf Seite 36)

[jerh] http://jersey.java.net/nonav/documentation/snapshot/jaxrs-resources.html#d4e146. (Zitiert auf Seite 36)

[jeri] http://jersey.java.net/nonav/apidocs/1.5/jersey/javax/ws/rs/core/Response.ResponseBuilder.html. (Zitiert auf Seite 36)

[jerj] http://jersey.java.net/nonav/apidocs/1.4/jersey/com/sun/jersey/api/client/WebResource.html. (Zitiert auf Seite 36)

[kar] http://weblogs.java.net/blog/mkarg/archive/2010/02/14/what-hateoas-actually-means. (Zitiert auf Seite 22)

[lgp] GNU Lesser General Public License. URL http://www.gnu.org/licenses/lgpl.html. (Zitiert auf Seite 41)

[LTB12] J. Louvel, T. Templier, T. Boileau. *Restlet in Action: Developing RESTful Web APIs in Java*. Running Series. Manning Publications Company, 2012. URL http://books.google.de/books?id=e14GywAACAAJ. (Zitiert auf den Seiten 95 und 96)

[plaa] http://www.playframework.com/documentation/1.2/libs. (Zitiert auf Seite 106)

[plab] http://www.playframework.com/documentation/2.1.1/JavaContentNegotiation. (Zitiert auf Seite 110)

[plac] http://www.playframework.com/documentation/1.2/routes#content-types. (Zitiert auf Seite 110)

[plad] http://www.playframework.com/documentation/1.2/routes#content-negotiation. (Zitiert auf Seite 110)

[Plae] http://www.playframework.com/documentation/2.0/JavaEbean. (Zitiert auf Seite 112)

[plaf] http://www.playframework.com/documentation/1.2.3/jpa. (Zitiert auf Seite 112)

[plag] https://github.com/ransombriggs/play-1.2.5-patched/blob/master/documentation/manual/production.textile#https-configuration. (Zitiert auf Seite 112)

[plah] http://www.playframework.com/documentation/1.2.3/configuration#http. (Zitiert auf Seite 112)

[plai] http://www.playframework.com/documentation/1.2.4/guide8. (Zitiert auf Seite 112)

[plaj] http://www.playframework.com/documentation/1.2.4/configuration#application. (Zitiert auf Seite 112)

[plak] http://www.playframework.com/documentation/2.1.1/JavaAsync. (Zitiert auf Seite 113)

[plal] http://www.playframework.com/documentation/1.2/asynchronous. (Zitiert auf Seite 113)

[plam] http://www.playframework.com/documentation/2.0/JavaStream. (Zitiert auf Seite 113)

[plan] http://www.playframework.com/documentation/2.1.1/JavaFileUpload. (Zitiert auf Seite 113)

[plao] http://www.playframework.com/documentation/2.0/JavaBodyParsers. (Zitiert auf Seite 113)

[resa] "http://docs.jboss.org/resteasy/docs/2.3.6.Final/userguide/html/Installation_Configuration.html#d4e31". (Zitiert auf Seite 78)

[resb] http://docs.jboss.org/resteasy/docs/3.0-rc-1/userguide/html/Content_Marshalling_Providers.html#Default_Providers_and_default_JAX-RS_Content_Marshalling. (Zitiert auf Seite 82)

[resc] http://restlet.tigris.org/ds/viewForumSummary.do?dsForumId=4447. (Zitiert auf Seite 95)

[resd] https://github.com/restlet/restlet-framework-java. (Zitiert auf Seite 95)

[rese] http://restlet.org/learn/guide/2.1/introduction/first-steps/first-server. (Zitiert auf Seite 96)

[resf] http://restlet.org/learn/guide/2.1/core/resource/client. (Zitiert auf Seite 96)

[resg] http://restlet.org/learn/guide/2.2/extensions/wadl. (Zitiert auf Seite 97)

[resh] http://restlet.org/learn/javadocs/2.2/gae/api/org/restlet/data/MediaType.html. (Zitiert auf Seite 100)

[rfc] http://tools.ietf.org/html/rfc5988#section-5. (Zitiert auf Seite 35)

[scoa] https://groups.google.com/forum/?fromgroups#!forum/scooter-framework. (Zitiert auf Seite 41)

[scob] https://twitter.com/scooterflies. (Zitiert auf Seite 41)

[scoc] http://www.scooterframework.com/docs/restful_routing.html. (Zitiert auf den Seiten 42 und 44)

[scod] http://www.scooterframework.com/docs/content_handler.html. (Zitiert auf Seite 45)

[scoe] http://www.scooterframework.com/docs/routes.html. (Zitiert auf Seite 46)

[scof] https://github.com/scooter/scooter/blob/master/source/src/com/scooterframework/web/route/RouteConstants.java. (Zitiert auf Seite 47)

[scog] http://www.scooterframework.com/docs/transactions.html. (Zitiert auf Seite 48)

[sum] https://java.net/projects/sommer. (Zitiert auf Seite 35)

[Tilo09] S. Tilkov. *REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien*. dpunkt, Heidelberg, 2009. (Zitiert auf den Seiten 20, 21, 22, 24 und 26)

[Vino08] S. Vinoski. RESTful Web Services Development Checklist. *Internet Computing, IEEE*, 12(6):96–95, 2008. doi:10.1109/MIC.2008.130. (Zitiert auf Seite 15)

[vraa] https://groups.google.com/forum/?fromgroups#!forum/caelum-vraptor-en. (Zitiert auf Seite 51)

[vrab] https://groups.google.com/forum/?fromgroups#!forum/caelum-vraptor-dev. (Zitiert auf Seite 51)

[vrac] https://github.com/caelum/vraptor. (Zitiert auf Seite 51)

[vrad] https://github.com/caelum/vraptor-contrib. (Zitiert auf Seite 51)

[vrae] https://groups.google.com/forum/?fromgroups=#!topic/restfulie-java/FiKg4ql8Dxs. (Zitiert auf Seite 51)

[vraf] https://github.com/caelum/vraptor/blob/master/vraptor-core/pom.xml. (Zitiert auf Seite 51)

[vrag] http://vraptor.caelum.com.br/en/docs/resources-rest-en/. (Zitiert auf Seite 53)

[vrah] https://github.com/caelum/vraptor-dash/blob/master/src/main/java/br/com/caelum/vraptor/dash/cache/ObservableResponse.java. (Zitiert auf Seite 55)

[vrai] http://www.guj.com.br/java/233197-vraptor-e-https. (Zitiert auf Seite 57)

[vraj] https://github.com/qmx/vraptor-authz. (Zitiert auf Seite 57)

[vrak] https://github.com/rlazoti/vraptor-authentication. (Zitiert auf Seite 57)

[vral] http://vraptor.caelum.com.br/en/docs/validation/. (Zitiert auf Seite 58)

[vram] http://vraptor.caelum.com.br/en/docs/download-and-upload/. (Zitiert auf Seite 58)

[win] http://wink.apache.org/. (Zitiert auf Seite 86)

[ZBD11] I. Zuzak, I. Budiselic, G. Delac. A finite-state machine approach for modeling and analyzing restful systems. *J. Web Eng.*, 10(4):353–390, 2011. URL http://dl.acm.org/citation.cfm?id=2230856.2230859. (Zitiert auf den Seiten 14, 22 und 121)

[ZS12] I. Zuzak, S. Schreier. ArRESTed Development: Guidelines for Designing REST Frameworks. *IEEE Internet Computing*, 16(4):26–35, 2012. doi:http://doi.ieeecomputersociety.org/10.1109/MIC.2012.60. (Zitiert auf den Seiten 14, 21 und 22)

[zus] http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven. (Zitiert auf Seite 121)

Alle URLs wurden zuletzt am 16. 06. 2013 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift