Ostfalia - Hochschule für angewandte Wissenschaften
Fachbereich Wirtschaft
Studiengang Wirtschaftsinformatik

Vergleich von Hypermedia Formaten

Bachelorarbeit

Zur Erlangung des Grades eines Bachelor of Science
der Fakultät Wirtschaft
der Ostfalia - Hochschule für angewandte Wissenschaften

eingereicht bei Dipl. Inf. Alex
Prof. Dr. Ey

von Marius Brederlow
Pappelweg 1
38536 Meinersen OT Ahnsen
Mat.-Nr. 70047338

Meinersen, den 12. August 2013

Zusammenfassung

Die vorliegende Arbeit befasst sich mit dem Vergleich verschiedener Hypermedia-Formate. Der Vergleich bezieht sich hierbei stets auf den Einsatz von Hypermedia-Nachrichten in einem verteilten Informationssystem. Um unterschiedliche Formate miteinander vergleichbar zu machen, wird eine theoretische Grundlage beschrieben und mehrere Faktoren zu Vergleichszwecken verwendet. Der Vergleich gibt im Wesentlichen Auskunft darüber, wie ein bestimmtes Format in einer verteilten Anwendungsarchitektur zum Einsatz kommen kann und diese unterstützt. Veranschaulicht wird dies, durch die Einführung eines Fallbeispiels. Die Evaluation der einzelnen Faktoren beschreibt die Fähigkeiten und Eigenschaften des betrachteten Formats. Der Einsatz von Hypermedia soll in zukünftigen Anwendungen zu einer guten Skalierbarkeit und einer robusten und erweiterbaren Architektur führen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CSV	Comma-separated Values
HAL	Hypermedia Application Language
HTTP	Hypertext Transfer Protocol
IANA	Internet assigned Numbers Authority
IETF	Internet Engineering Task Force
JPEG	Joint Photographic Expert Group
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
REST	Representational State Transfer
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
URI	Unified Ressource Identifier
XHTML	Extensible Markup Language
XML	Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

1.1 Motivation und Zielsetzung

Wenn wir heutzutage über die Verwendung und Vorteile von Hypermedia sprechen, ist dies keine Innovation der letzten Jahre. Bereits im Jahr 1990 beschrieb Tim Berners-Lee in seinem Vorschlag für ein Hypertext-Projekt die wesentlichen Eigenschaften einer Hypertext basierten Anwendung:

"The current incompatibilities of the platforms and tools make it impossible to access existing information through a common interface, leading to waste of time, frustration and obsolete answers to simple data lookup. There is a potential large benefit from the integration of a variety of systems in a way which allows a user to follow links pointing from one piece of information to another one. This forming of a web of information nodes rather than a hierarchical tree or an ordered list is the basic concept behind HyperText."[1]

Die Idee von Berners-Lee ist eine Anwendung nicht in Form von sequentiellen Abläufen und einer hierarchischen Struktur abzubilden, sondern als eine Art Sammlung von lose gekoppelten Ressourcen mit der Möglichkeit sich in dieser Sammlung vor und zurück zu bewegen. Der Vorschlag bewegt sich damit weg von den hierarchischen Strukturen einer verketteten Liste oder der Darstellung als Baum. Diese Idee legte seinerzeit den Grundstein für das, was wir heute als das WorldWideWeb kennen. Das Internet ist wohl das größte und umfassendste, verteilte Informationssystem unserer Zeit. Der Erfolg des Internet beruht sicherlich auf der Tatsache, dass es möglich ist verschiedenste Arten von Informationen rund um den Globus miteinander zu vernetzen und abrufbar zu machen.
Im Jahr 2000 hat sich Roy Fielding im Rahmen seiner Dissertation mit dem Thema beschäftigt, welche Architektur das Internet selbst besitzt und wie diese Architektur aussehen sollte. Das Internet ist für ihn per Definition ein verteiltes Hypermedia-System, wobei er Hypermedia als eine Weiterentwicklung von Hypertext betrachtet. Hypermedia umfasst im Gegensatz zu Hypertext nicht nur Verweise auf anderen Hypertext, sondern auf multimediale Inhalte wie Text, Audio und Video. Fielding selbst schlägt als Architektur einer verteilten Hypermedia Anwendung, das von ihm beschriebene "Representational State Transfer"(REST) vor. Ein zentraler Bestandteil dieser Architektur ist die Repräsentation von Ressourcen und die Steuerung der Anwendung durch den Einsatz von Hypermedia.
Dieser Architekturstil gewann in den letzten Jahren zunehmend an Bedeutung bei der Erstellung von verteilten Hypermedia-Systemen. Mensch-zu-Maschine Kommunikation und reine Maschinenkommunikation kann gleichermaßen abgebildet werden. Eine Idee beim Einsatz von Hypermedia in verteilten Informationssystemen besteht darin, das Verhalten beim Browsen durch

[1] Berners-Lee, T. (1990), `http://www.w3.org/Proposal.html`

das Internet auch außerhalb des Browsers anwendbar zu machen. Bei der Verwendung eines Browsers kann ein Mensch anhand von Beschreibungen oder dem Kontext der Information einem Verweis folgen. Das Folgen der Verweise ist nicht sequentiell, sondern steht immer im Kontext der eingebetteten Informationen. Mit anderen Worten, ein Mensch kann die Semantik der Informationen verstehen und dem gewünschten Link folgen. Dies ist möglich ohne den exakten Endpunkt des Verweises zu kennen. An dieser Stelle spiegelt sich auch ein wesentlicher Erfolgsfaktor des Internets bzw. der Nutzung von Hypertext wieder. Niemand kann sich eine große Anzahl von URIs (Uniform Ressource Identifier) merken. Selbst wenn jemand sich alle wichtigen URIs merken könnte ist nicht sichergestellt, dass sie über die Zeit stabil sind. Stabilität ist auch nicht notwendig, da das Folgen eines Links ohne den exakten Endpunkt zu kennen möglich ist.
Betrachtet man eine Client-Server-Architektur als einen oft eigesetzten Stil für verteilte Anwendungen, muss der Client seine verwendeten Endpunkte auf den Servern kennen. Bei vielen Client Applikationen sind diese Endpunkte als ein Bestandteil des Quellcodes festgelegt. Diese Tatsache hat zur Folge, dass im Fall einer Änderung oder Erweiterung des Adressraums die im Client festgelegten URIs invalidiert werden. Demnach besteht schon aufgrund der statischen Benutzung von URIs eine recht enge Kopplung zwischen Client und Server. Um einen lose gekoppelten Client zu entwerfen, sollte er keine festgelegten URIs aus dem Adress-Schema des Servers enthalten. Die einzige Ausnahme ist hier ein zentraler Eintrittspunkt, von dem alle weiterführenden Aktionen ausgehen. Die weiteren Operationsmöglichkeiten und die Navigation durch die Anwendung sollte vom Server dynamisch zur Laufzeit an den Client gesendet werden.
Dieses Vorgehen ist bei einer Mensch-zu-Maschine Kommunikation intuitiv umsetzbar. Es entspricht exakt der Anwendung eines Browsers um durch verschiedene Webseiten als Repräsentation von Informationen im Internet zu navigieren. Intuitive Benutzbarkeit kommt zu stande, weil ein Mensch in der Lage ist den Kontext der Information zu interpretieren und entsprechend zu handeln.
Bei einer Maschine-zu-Maschine Kommunikation ist das nicht der Fall. Eine Maschine kann in der Lage sein einem Verweis zu folgen, sie kennt jedoch nicht die Semantik des Verweises. Hierfür ist zusätzliches Wissen auf der Seite des Clients erforderlich. Bei vielen Anwendungen ist dieses Wissen in einer umfassenden Dokumentation festgelegt. Die Dokumentation beschreibt, welche Verweise und Endpunkte für eine bestimmte Anwendungslogik zur Verfügung stehen. Client-Applikationen müssen dieses Wissen implementieren um die Nachrichten der Anwendung zu verstehen und die Anwendung zu steuern. Lose gekoppelte Systeme sind nach diesem Modell nur schwer zu entwerfen. Änderungen an den verwendeten Adressen müssen im Client stets reimplementiert werden.
Hypermedia setzt an dieser Stelle auf ein anderes Prinzip. Um die Implementierung von Wissen auf der Clientseite möglichst gering zu halten, werden selbstbeschreibende Hypermedia-Nachrichten eingesetzt. Selbstbeschreibende Nachrichten sind angereichert mit semantischen Informationen für die Verarbeitung der Nachricht. Also mit dem Wissen das der Client benötigt um die erhaltenen Informationen darzustellen und die Anwendung zu steuern. Sämtliches, in einer externen Dokumentation festgelegtes Wissen, das nicht als ein Bestandteil der selbstbeschrei-

benden Nachricht mitgeliefert wird, ist im Hypermedia Umfeld als 'Out of Band Knowledge' bezeichnet. Der Anteil dieser Wissensform sollte im Allgemeinen möglichst gering gehalten werden, um eine zu starke Kopplung zu vermeiden.
Reduziert man eine verteilte Hypermedia-Anwendung auf das wesentliche Vorgehen, ergeben sich zwei charakteristische Eigenschaften. Auf der einen Seite ein Client, welcher in der Lage ist, Informationen darzustellen und Verweisen zu folgen. Auf der anderen Seite ein Server, der Informationen und Verweise auf seine Ressourcen zur Verfügung stellt. Als ein zentraler Bestandteil der Kommunikation können somit Verweise identifiziert werden. Bestandteile eines Verweises selbst lassen sich anhand der genannten Eigenschaften einer verteilten Hypermedia-Anwendung abstrakt in drei Attribute unterteilen:

- Der URI als aktueller vom Server zur Verfügung gestellter Endpunkt für eine Ressource
- Das Relationsattribut zur Beschreibung der Beziehung zwischen aktuell genutzter und Zielressource
- Das Methoden Attribut um festzulegen mit welcher Protokollmethode die Ressource angesprochen werden kann

Verweise, angereichert mit der beschriebenen Semantik, lösen die Kopplung zwischen Client und Server bis zu einem gewissen Grad. Clients einer verteilten Hypermedia Anwendung benötigen im Idealfall nur Kenntnis über die verwendeten Relationsattribute, wobei unterstellt wird, dass der Client den zentralen Eintrittspunkt der Anwendung kennt und der Server seine Adressen komplett selbständig verwalten kann. Die Steuerung der Anwendung erfolgt ausschließlich über die vom Server zur Verfügung gestellten Verweise. Dieses Vorgehen entspricht dem eines Browsers und einer Hypermedia getriebenen Anwendung gleichermaßen.
Hypermedia hat viel Potential in Hinblick auf die Erstellung eines verteilten Informationssystems. Durch semantisch angereicherte Nachrichten wird versucht die Kopplung zwischen Client und Server auf ein Minimum zu reduzieren. Ein her geht damit auch eine bessere Wartbarkeit und Erweiterbarkeit der Applikation. Essentieller Grundbestandteil der Kommunikation ist ein Hypermediaformat, das möglichst viele der geforderten Eigenschaften unterstützt und einen lose gekoppelten Nachrichtenaustausch ermöglicht. Aktuell existieren viele verschiedene Formate, die den Erfordernissen in unterschiedlichen Umfängen genügen. Diese Arbeit hat zum Ziel verschiedene Formate miteinander zu vergleichen und auf ihre Leistungsfähigkeit hin zu untersuchen.

2 Hypermedia

2.1 Definition

Das geläufigste Format von Hypermedia ist HTML in Verbindung mit dem Internet und Millionen von Websites. Hypermedia auf Websites zeichnet sich auf den ersten Blick durch zwei wesentliche Merkmale aus. Websites bestehen zum einen aus multimedialen Inhalten und sind zum anderen untereinander durch Verweise verknüpft. Außerdem gibt es die Möglichkeit Daten an den Server über Formulare zu senden. Diese Merkmale lassen sich dem bekanntesten Hypermediaformat, (X)HTML ohne weiteres zuschreiben.
Roy Fielding beschreibt im Rahmen seiner Dissertation REST als einen Architekturstil, auf Grundlage des modernen Internets. Fielding definiert das Internet hierbei als ein hochgradig verteiltes Hypermedia-System, dessen Verteilung über die geographische Ausdehnung hinaus geht und eine grenzenlose Verbindung von Informationen ermöglicht. Dienstanbieter müssen mit einer unabhängigen Entwicklung von Software-Komponenten, sowie der unregulierten Skalierung des Internet zurecht kommen. Als Schlüsseltechnologie kommt an dieser Stelle Hypermedia zum Einsatz. Hypermedia ist im Rahmen des Internet ein einheitliches Mittel um auf Dienste zuzugreifen. Der Zugriff erfolgt über eingebettete Steuerelemente als ein Teil der erhaltenen Informationen.[2]
Hypermedia bildet zudem eine universell einsetzbare Schnittstelle im Internet. Die gleiche Schnittstelle kann unabhängig von verschiedenen Informationsquellen genutzt werden. Verweise erlauben eine unbegrenzte Strukturierung der Informationen und die Manipulation der Verweise ermöglicht einem Dienstanbieter die Darstellung komplexer Beziehungen zwischen Informationen. Auf diese Weise kann ein Dienstkonsument durch die Anwendung geleitet werden. Außerdem können einfache Anfragen durchgeführt werden, die das Durchsehen von großen Datenbeständen vereinfachen. Einfachheit und Allgemeingültigkeit lassen Hypermedia zu einer leicht zugänglichen Technologie werden.[3]
REST abstrahiert von den implementativen Details und fokussiert den Einsatz von Hypermedia mit der Interpretation von signifikanten Datenelementen durch selbst beschreibende Nachrichten. Diese Hypermedia-Nachrichten bestehen aus sechs Datenelementen:

- Ressource
- Ressourcenidentifikator
- Repräsentation der Information
- Metadaten der Repräsentation
- Metadaten der Ressource

[2] Vgl. Fielding, R. (2000), S. 3
[3] Vgl. Fielding, R. (2000), S. 67f

- Steuerdaten

Dinge der realen Welt werden durch die Ressource repräsentiert. Die Ressource selbst stellt eine Abstraktion der Information dar und kann einen Dienst, ein Dokument oder eine Sammlung anderer Ressourcen beinhalten. Der Ressourcenidentifikator dient der Adressierung und Referenzierung einer Ressource. Eine stabile und semantisch korrekte Identifikation liegt hierbei in der Verantwortung der Ressourcen verwaltenden Komponente. Die Repräsentation einer Ressource besteht aus Nutzdaten und Metadaten der Repräsentation zu Beschreibungszwecken. Zusätzlich kann eine Antwort auch Metadaten der Ressource enthalten. Diese beinhalten Informationen der Ressource, die nicht spezifisch zu der Repräsentation zugeordnet werden können. Die Steuerdaten dienen der Parametrisierung einer Anfrage. Sie beschreiben die Bedeutung und Verwendung einer bestimmten Aktion innerhalb der Nachricht.[4]

"Hypermedia is defined by the presence of application control information embedded within, or as layer above, the presentation of information."[5]

Während des Entwurfs einer verteilten Hypermedia-Anwendung müssen diese Anforderungen durch den Einsatz eines bestimmten Datenformats abgebildet werden. Die Betrachtung von verschiedenen Hypermediaformaten soll innerhalb dieser Arbeit losgelöst von Programmiersprachen, Technologien und serverseitigen Implementationen erfolgen. Im Mittelpunkt steht eine Betrachtung der Formate entsprechend ihrer Eigenschaften und Fähigkeiten eine Hypermedia-Anwendung zu unterstützen.

2.2 Richardsons Maturity Model

Leonard Richardson beschreibt mit seinem in Abbildung 1 dargestellten Maturity Model eine Art Stufenplan, wie ein verteiltes Hypermedia System auf Basis von Internet Technologien umgesetzt werden kann. Die konsequente Umsetzung von REST als Architekturstil und das nutzen bestehender Technologien steht hierbei im Fokus. Das Zusammenwirken von HTTP, URIs und Hypermedia in Form von HTML ist charakteristisch für das Internet. Diese drei Technologien und dessen Kombination bilden die Grundlage für das Erstellen einer Hypermedia-Anwendung.[6]

[4] Vgl. Fielding, R. (2000), S. 86ff
[5] Fielding, R. (2000), S. 68
[6] Vgl. Richardson, L. (2008), `http://www.crummy.com/writing/speaking/2008-QCon/act3.html`

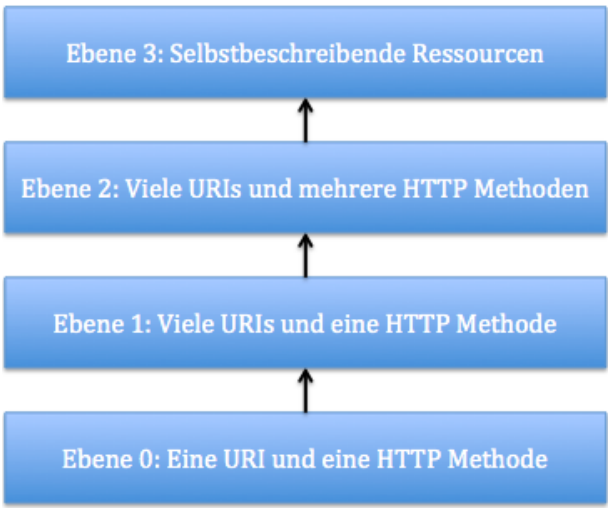

Abbildung 1: Leonard Richardsons Maturity Model

Um die Unterschiede zwischen den einzelnen Ebenen darzustellen, wird ein sehr einfaches Warenwirtschaftssystem eingeführt. Dieses System bietet die Möglichkeit, eine Liste des gesamten Warenbestands anzufragen, die Details einer bestimmten Ware anzuzeigen und Bestellungen von Waren auszulösen.

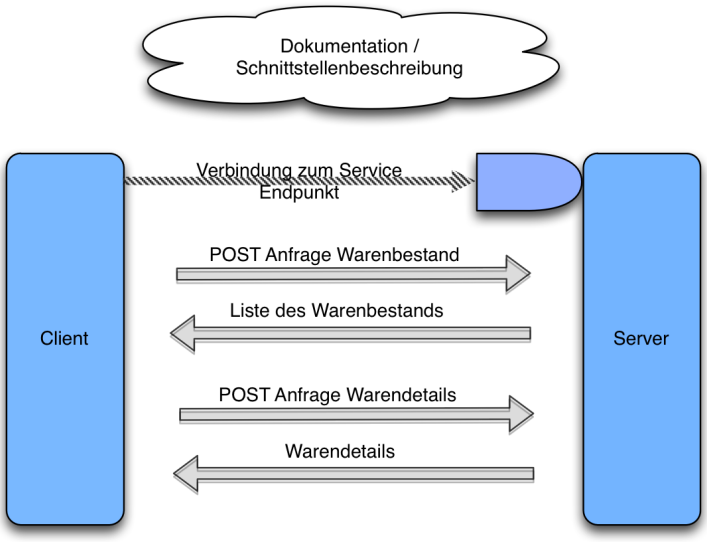

Abbildung 2: Maturity Model Ebene 0

Abbildung 2 zeigt, wie eine Umsetzung des Systems auf Ebene 0 aussehen kann. Auf dieser Ebene sind alle Funktionen außerhalb der Nachrichten definiert und werden in der Regel durch ein HTTP-POST getunnelt. HTTP dient auf dieser Ebene als Intermediär und löst selbst implementiere, entfernte Prozeduren oder Methoden aus. Außer POST werden keine anderen Verben ver-

wendet. Somit wird keine semantisch korrekte Verwendung von HTTP umgesetzt und POST kann auch für lesende Operationen verwendet werden. Ein Rückschluss vom POST auf die tatsächlich aufgerufene Methode oder Prozedur ist nicht möglich. Diese Vorgehensweise ähnelt stark dem Konzept der Remote Method Invocation im Fall von JAVA. Die Kommunikation basiert auf dem serialisieren und deserialisieren von Objekten. Sämtliches Wissen über die Objekte und deren Format ist in einer externen Dokumentation festgelegt. Ein Beispiel ist das Senden von XML Dateien. Das Verständnis über die ausgetauschten XML-Instanzen liegt innerhalb einer Dokumentation und XML-Schemata werden zur Validation eingesetzt. Dies ist auch der Fall beim 'Simple Object Access Protocol' (SOAP), der als ein industrieller Standard für den Austausch von Daten eingesetzt wird. Im Unterschied zu reinem XML wird die XML-Instanz hier jedoch in einen Umschlag verpackt. Die Dokumentation erfolgt in einer Beschreibungssprache und liegt somit ebenfalls in externer Form vor.[7]

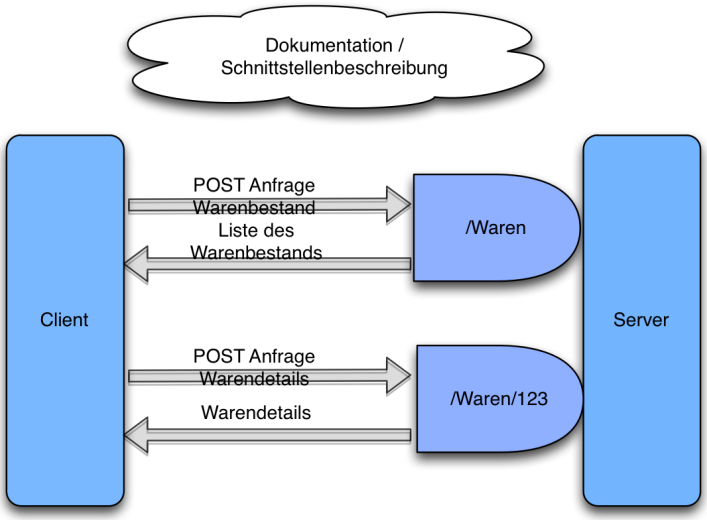

Abbildung 3: Maturity Model Ebene 1

Die Verwendung von Ressourcen kommt, wie in Abbildung 3 dargestellt, auf Ebene 1 zum Tragen. Anstatt alle Anfragen an einen zentralen Endpunkt zu richten, werden auf dieser Ebene Ressourcen direkt angesprochen. Das Konzept der Ressourcen ermöglicht somit eine Art Identifikation von Dingen der realen Welt. Sollten Details zu einer bestimmten Ware benötigt werden, kann diese nun direkt angesprochen werden. Alle Anfragen werden jedoch weiterhin nur durch eine HTTP-Methode abgebildet. Ein semantischer Rückschluss zwischen der genutzten HTTP-Methode und der konkret aufgerufenen Methode ist weiterhin nicht möglich.[8]

[7] Vgl. Richardson, L. (2008), `http://www.crummy.com/writing/speaking/2008-QCon/act3.html`
[8] Vgl. Richardson, L. (2008), `http://www.crummy.com/writing/speaking/2008-QCon/act3.html`

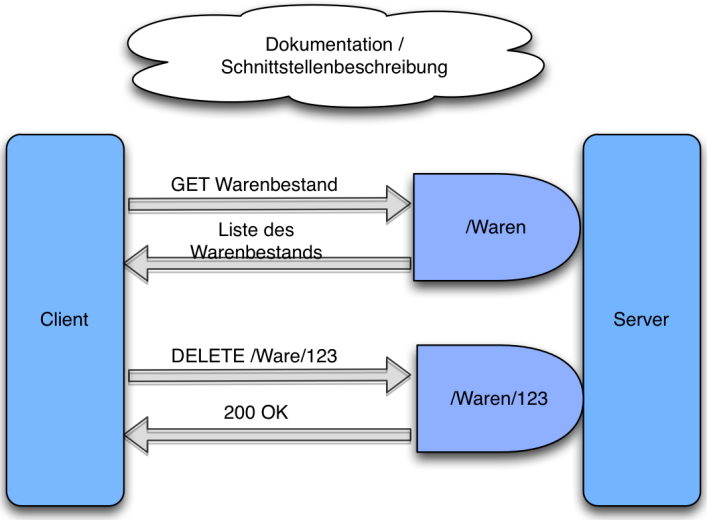

Abbildung 4: Maturity Model Ebene 2

Eine Unterscheidung zwischen lesenden, schreibenden, idempotenten und nicht idempotenten Operationen, konnte bis jetzt durch den Einsatz von HTTP als Tunnelmechanismus nicht vorgenommen werden. Eine entfernte Methode wurde stets durch ein POST ausgelöst. Die HTTP konforme Verwendung der Verben wird auf der in Abbildung 4 dargestellten Ebene 2 eingeführt. Anstatt die Liste des Warenbestands über eine POST-Anfrage auszuführen wird eine GET-Anfrage abgesetzt. GET ist im Rahmen von HTTP definiert als eine sichere idempotente Operation, die keine signifikanten Änderungen am Status der Anwendung durchführt. Die Abfrage des Warenbestandes muss aufgrund ihres lesenden Charakters und der Beibehaltung des Anwendungszustands ab dieser Ebene mittels einer GET-Anfrage durchgeführt werden. PUT, POST und DELETE sind innerhalb von HTTP definiert als Operationen mit schreibendem Charakter und demzufolge einer Änderung des Anwendungszustands. Wenn eine Ware geändert oder gelöscht werden soll, muss eine entsprechende Operation verwendet werden. Neben der korrekten Verwendung der Verben wird auf dieser Ebene ein Mechanismus zur Fehlerindikation eingesetzt. Die Verwendung der HTTP-Status Codes entsprechend ihrer Spezifikation muss diese Rolle übernehmen. So muss zum Beispiel die korrekte Abarbeitung einer Anfrage vom Dienstanbieter mit einem 2xx quittiert werden. Die Umsetzung eines geeigneten Fehlerindikators liegt in der Verantwortung des Dienstanbieters.[9]

[9] Vgl. Richardson, L. (2008), `http://www.crummy.com/writing/speaking/2008-QCon/act3.html`

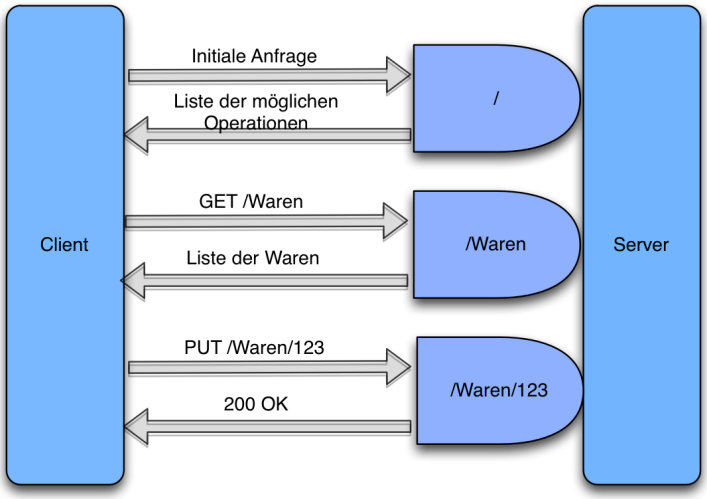

Abbildung 5: Maturity Model Ebene 3

Abbildung 5 zeigt das System auf Ebene 3 des Maturity Modells. Hier wird mit der Verwendung von Hypermedia-Steuerelementen begonnen. Eine Nachricht auf dieser Ebene enthält nicht nur reine Nutzdaten, sondern auch semantische Informationen zum Verständnis der Daten. Hypermedia Steuerelemente zeigen einem Client welche Operationen er von seiner aktuellen Position aus durchführen kann. Der Client benötigt keine Kenntnis über die exakten Adressen für bestimmte Operationen. Vielmehr wird er vom einem zentralen Einstiegspunkt per Verweis durch den Anwendungsprozess geführt. Diese selbstbeschreibenden Nachrichten ermöglichen einen variablen Adressraum auf der Seite des Dienstanbieters und reduzieren die Kopplung zwischen Client und Server. Zudem kann der Dienstanbieter neue Verweise und Informationen zu seinen Nachrichten hinzufügen. Solange alle Dienstkonsumenten unbekannte Verweise ignorieren, bietet der Einsatz von Hypermedia-Steuerelementen eine gefahrlose Erweiterungsmöglichkeit der Nachrichten. Wenn das Warenwirtschaftssystem auf dieser Ebene operiert, benötigt der Client als Information nur den zentralen Einstiegspunkt und Kenntnis über die genutzten Verweise. Im Idealfall ist keine externe Dokumentation mehr nötig. Der Client kennt die benutzten Verweise und weiß anhand der selbstbeschreibenden Nachrichten, mit welchen Operationen er auf eine Ressource zugreifen kann.
Das Richardson Maturity Model bietet keinen direkten Ansatz zur Bewertung eines bestimmten Hypermedia-Formats. Es beschreibt jedoch Eigenschaften, die für das Erstellen einer REST konformen Anwendung und damit einer Hypermedia-Anwendung nötig sind. Besonders die Erfordernisse der dritten Ebene beschreiben Eigenschaften, die im Einflussbereich verschiedener Hypermedia-Formate liegen. Eine detaillierte Betrachtung des eingesetzten Formats ist nötig, um eine gute Anwendung zu erstellen und die gegebenen Vorteile im vollen Umfang zu nutzen. Dieser Anspruch impliziert die hohe Bedeutung für das Erstellen einer Vergleichsgrundlage für verschiedene Formate.[10]

[10] Vgl. Richardson, L. (2008), `http://www.crummy.com/writing/speaking/2008-QCon/act3.html`

2.3 Hypermedia Faktoren nach Amundsen

Der Hypermediagedanke entstand aus der Fragestellung, wie man ein gutes verteiltes Informationssystem entwerfen kann. Grundsätzlich gibt es verschiedene Ansätze um ein verteiltes System zu entwerfen und die Komponenten miteinander kommunizieren zu lassen. Die zu grunde liegende Fragestellung ist hier, wie kann der Server private Objekte exportieren um sie für den Client sichtbar und benutzbar zu machen. Hierfür gibt es unterschiedliche plattform- und sprachspezifische Lösungen. Die grundsätzliche Funktionsweise ist hierbei immer ähnlich. Die Objekte werden serialisiert über das Netzwerk geschickt und so an den Client zur Verarbeitung übergeben. Der Client selbst muss also über ein Verständnis der Daten verfügen. Diese Herangehensweise hat den Nachteil, dass eine Änderung der serialisierbaren Objekte auch eine Änderung des Clients zur Folge hat. Somit entsteht immer ein gewisser Grad an Kopplung zwischen Client und Server.
Amundsen beschreibt mit Hypermedia einen anderen Lösungsansatz. Eine Synchronisation von Client und Server, bei der private Datentypen miteinander geteilt werden, ist nicht der richtige Ansatz. Stattdessen sollte eine Technik zur Datenbeschreibung unabhängig von Plattform, Sprache und privaten Datentypen verwendet werden. Diese Herangehensweise nennt Amundsen die Hypermedia-Lösung. Die Probleme des Typemarshalling werden im Fall von Hypermedia durch das Anreichern der Nachrichten mit zwei Arten von Metadaten gelöst. Ein Teil sind Metadaten über die Daten selbst. Der andere Teil sind Metadaten über den Applikationsstatus und die Möglichkeiten für den Client, den Applikationsstatus zu ändern. Losgelöst von privaten Datentypen ermöglicht dieses nachrichtenorientierte Design, eine spezifischere Anpassung an die Anforderungen der Problemdomäne. Durch Hypermedia-Nachrichten werden somit keine privaten Datentypen geteilt, sondern vielmehr ein generelles Verständnis der Daten.[11]
Nach Amundsen besteht eine Hypermedia-Nachricht somit aus insgesamt drei Teilen:

- Daten
- Metadaten über die Daten
- Metadaten über den Applikationsstatus

In der Regel wird bei Hypermedia-Applikationen HTTP als Transportprotokoll eingesetzt, weil es eine Reihe von Vorteilen wie z.B. Caching, Chunking und Kompression bereits implementiert hat. HTTP ist aber keine zwingende Vorraussetzung für den Einsatz von Hypermedia. Vielmehr sollte die Unabhängigkeit vom Transportprotokoll stets gewahrt werden.
Um Hypermedia selbst zu identifizieren, definiert Amundsen eine Sammlung abstrakter Eigenschaften und Funktionen, wie ein Hypermedia-Format diese drei charakteristischen Datenteile umsetzt. Diese Sammlung wird Hypermedia-Faktoren (H-Faktoren) genannt und ist grundsätzlich unabhängig von der verwendeten Plattform und dem Transportprotokoll. H-Faktoren unterteilen sich dann noch in die zwei Bereiche Links und Control Data. Diese insgesamt neun H-Faktoren

[11] Vgl. Amundsen, M. (2012), S. 10

bilden das Gerüst für ein beliebiges Hypermedia-Format. So kann man anhand dieser Faktoren bestimmen, ob ein bereits existierender Medientyp den applikationsspezifischen Anforderungen genügt oder aber was bei der Erstellung eines eigenen Medientyps beachtet werden muss.[12]
Im Folgenden werden die einzelnen Faktoren genauer betrachtet. Die Link-Faktoren dienen im Wesentlichen der Bewegung des Clients im Applikationsfluss, wohingegen die Control-Faktoren dadurch definiert sind, dass sie zusätzliche Informationen beim Ausführen eines Verweises zur Verfügung stellen. Hier geht es vor allem um die Beschreibung der beinhalteten Nutzdaten. Die Berücksichtigung dieser Faktoren trägt dazu bei, dass alle Seiten etwas über die Beschaffenheit der Daten wissen oder aber die Daten in einer gewünschten Kodierung anfordern können. Die Illustration der H-Faktoren erfolgt am Beispiel von HTTP mit HTML als eingesetzten Hypermedia-Typ. Die nachstehende Abbildung 6 gibt einen Überblick der betrachteten Hypermedia-Faktoren.

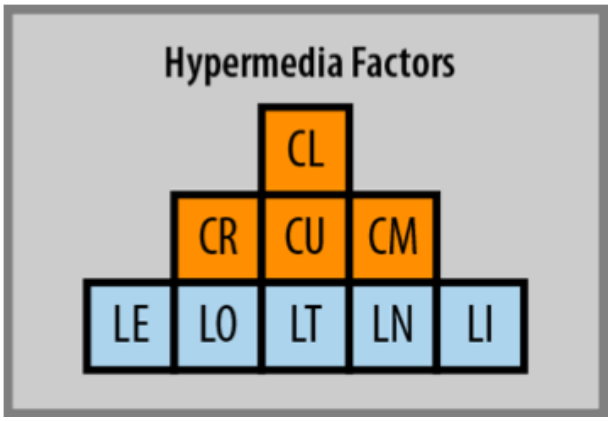

Abbildung 6: Hypermedia-Faktoren nach Amundsen

2.3.1 Embedding-Links

Embedding-Links (H-Faktor LE) dient der Darstellung einer Ressource im aktuell genutzten Ausgabefenster. Ein Embedding-Link nutzt die Lesen-Operation des verwendeten Protokolls und stellt die Antwort innerhalb des aktuellen Fensters dar. Ein gebräuchliches Beispiel für Embedding-Links sind auf einer Website eingebundene Elemente.[13]

Listing 1: HTML Beispiel LE

```
<iframe src="..."> Einbetten eines Rahmens
<img src="..."> Einbetten eines Bildes
```

[12] Vgl. Amundsen, M. (2012), S. 13f
[13] Vgl. Amundsen, M. (2012), S. 15

Die im Listing 1 dargestellten Tags sorgen bei der Darstellung im Browser für eine Einbettung von lokalen oder entfernten Ressourcen und sind im Fall von HTML repräsentativ für den H-Faktor der Embedding-Links.

2.3.2 Outbound-Links

Outbound-Links (H-Faktor LO) werden im Allgemeinen als Navigationslinks verstanden. Beim Folgen eines Outbound-Links wird die Lesen-Operation des verwendeten Protokolls ausgeführt und die bestehende Ansicht durch die Antwort ersetzt.[14] Die Antwort von Google auf eine Suchanfrage beinhaltet Outbound-Links. Folgt man einem Verweis aus der Antwort wird die darauf folgende Antwort die aktuelle Ansicht ersetzen.

Listing 2: HTML Beispiel LO

```
<a href="http://externe-domain.de">
```

In HTML werden Verweise durch das Anchor-Element (a) kenntlich gemacht. Die Kombination des Anchor- und Hyper-Reference- Elements (a href) sorgen beim Aktivieren für eine Darstellung der refferenzierten Ressource in einer neuen Ansicht.

2.3.3 Templated-Links

Templated-Links (H-Faktor LT) haben wie Embedding- und Outbound-Links nur einen lesenden Charakter. Templated-Links ermöglichen dem Client das Senden von Daten an einen Server. Beispielsweise können bei einem Suchdienst die Parameter übergeben werden, sodass die Suchbegriffe als ein Teil des Links mit übermittelt werden. Die Zusammensetzung des Links ist abhängig von der Spezifikation des genutzten Medientyps. Ein Verweis, der eine Suche durchführt, sollte entsprechend der Spezifikation wie im folgendem Listing zusammengesetzt sein.[15]

Listing 3: HTML Beispiel LT

```
<form action="http://domain.de" method="get">
 <input type="text" name="search" value=""/>
  <input type="submit"/>
</form>
```

Templated-Links für Suchanfragen in HTML bestehen nach Spezifikation aus der Zieldomain und einem Platzhalter für die vom Nutzer übergebene Zeichenkette.

[14] Vgl. Amundsen, M. (2012), S. 15
[15] Vgl. Amundsen, M. (2012), S. 16

2.3.4 Idempotent-Links

Idempotent-Links (H-Faktor LI) haben einen schreibenden Charakter und stellen eine Möglichkeit dar, Ressourcen auf dem Server anzulegen oder zu löschen. Idempotenz hängt hierbei natürlich von dem verwendeten Transportprotokoll und der serverseitigen Implementation ab. Bei der Verwendung von HTTP als Transportprotokoll sind GET, PUT und DELETE idempotente Operationen zur Kommunikation mit dem Server.[16]

Listing 4: HTML Beispiel LI

```
<script type="text/javascript">
  function update(ressource){
  var req = new XMLHttpRequest();
  req.open("PUT", "http://domain.de/ressource");
  req.send();
  }
</script>
```

Der im Listing 4 dargestellte Code dient dem Anlegen oder Aktualisieren einer Ressource. HTML selbst bietet keine direkte Unterstützung für idempotente Operationen. Ein in HTML eingebettetes Script kann jedoch einen idempotenten Request erzeugen und absenden. Im Zusammenspiel von HTML und HTTP als genutztes Transportprotokoll ist PUT als eine idempotente Operation implementiert und führt auch bei mehrfacher Ausführung immer zum gleichen Ergebnis.

2.3.5 Non-Idempotent-Links

Non-Idempotent-Links (H-Faktor LN) bietet wie H-Faktor LI eine Möglichkeit Daten zu übertragen. Der Unterschied hier ist die Idempotenz. So liegt es in der Verantwortung des Medientyps zu definieren, wie mit nicht idempotenten Operationen umgegangen wird. Im Fall von HTTP ist die Methode POST als nicht idempotente Operation implementiert. POST wird genutzt um per Formular eine neue Ressource auf dem Server anzulegen.[17]

Listing 5: HTML Beispiel LI

```
<form action="http://domain.de/ressource" method="post">
  <input type="submit"/>
</form>
```

[16] Vgl. Amundsen, M. (2012), S. 16
[17] Vgl. Amundsen, M. (2012), S. 17

HTML-Formulare, die als Operation POST nutzen um Daten zum Server zu senden, legen eine neue Ressource auf dem Server an. Im Gegensatz zu PUT als idempotente Operation liefert POST nach der Durchführung nicht immer das gleiche Ergebnis. Stattdessen legt POST bei mehrfacher Ausführung, mehrere neue Ressource an. Der Umgang mit nicht idempotenten Operationen liegt in der Verantwortung und Implementation des konkret verwendeten Medientyps.

2.3.6 Read-Controls

Read-Controls (H-Faktor CR) beschreibt ob und wie die Möglichkeit besteht, eine lesende Operation zu reglementieren. Ein Beispiel hierfür sind die Accept-Header des HTTP-Protokolls. Dem Client wird so die Möglichkeit gegeben ein bestimmtes Format oder die Antwort in einer bevorzugten Sprache anzufordern.[18]

Listing 6: HTML Beispiel CR

```
GET /beispiel/ressource HTTP/1.1
Host:      domain.de
Accept:  text/html
```

Read-Control wird an dieser Stelle durch einen HTTP-Header veranschaulicht. Der Header resultiert aus einer lesenden Anfrage auf eine Ressource. Das Accept-Element im Header setzt hierbei die Steuerung der lesenden Operation GET um und schränkt die Annahme auf das Format HTML ein. Ressourcen mit mehreren Repräsentationsformen können auf diese Weise in einem spezifiziertem und vom Client unterstützen Format angefordert werden.

2.3.7 Update-Controls

Update-Controls (H-Faktor CU) bietet die gleichen reglementierenden Möglichkeiten, die der H-Faktor CR für schreibende Operationen bietet. Die Reglementierung von schreibenden Operationen kann bei HTTP z.B. durch das Content-Typ Feld im Header durchgeführt werden. So hat der Server Kenntnis darüber, in welcher Form ihm die Nutzdaten übergeben werden.[19]

Listing 7: HTML Beispiel CU

```
<form action="ressource/bild">
  <input type="file" accept="image/jpeg">
  <input type="submit">
```

[18] Vgl. Amundsen, M. (2012), S. 18
[19] Vgl. Amundsen, M. (2012), S. 18

```
</form>
```

Durch das Accept-Attribut wird der Upload einer Datei beschränkt. Das Festlegen auf den Multipurpose Internet Mail Extension Type (MIME-Type) 'image/jpeg' stellt sicher, dass eine neue Ressource, die über dieses Formular erstellt wird, den MIME-Type JPEG haben muss. Ein Anlegen von neuen Ressourcen kann somit exakt reglementiert werden. Elementarer Bestandteil der Reglementierung ist das Verwenden von MIME-Types. Eine Liste aller offiziell registrierten MIME-Types wird durch die Organisation 'Internet Assigned Numbers Authority' (IANA) gepflegt und veröffentlicht.

2.3.8 Method-Controls

Method-Controls (H-Faktor CM) stellen Indikatoren dar, die beschreiben um was für eine Art von Operation es sich handelt bzw. welche Operation für einen konkreten Fall gültig sind. Ein HTML FORM-Element kann mit unterschiedlichen Methoden abgesendet werden. Das Attribut METHOD beschreibt an dieser Stelle mit welcher Operation des Transportprotokolls das Formular übertragen wird.[20]

Listing 8: HTML Beispiel CM

```
<form action="ressource/bild" method="get">
<input type="submit"/>
</form>

<form action="ressource/bild" method="post">
<input type="submit"/>
</form>
```

Beide dargestellten Formulare senden ihre Nutzdaten an '/ressource/bild'. Sie unterscheiden sich jedoch in der verwendeten Protokolloperation. Durch die Method-Controls kann am Beispiel von HTML festgelegt werden ob eine Ressource mit einer idempotenten oder mit einer nicht-idempotenten Operation angesprochen wird.

2.3.9 Link-Annotation-Controls

Zusätzlich zur Reglementierung von Schreib- und Leseoperationen kann ein Hypermedia-Typ Link-Annotation-Controls (CL) unterstützen. Verweise werden dann mit semantischen Informa-

[20] Vgl. Amundsen, M. (2012), S. 18f

tionen angereichert. Clients, die diese Metadaten verstehen, können sich entsprechend der Semantik verhalten.[21]

Listing 9: HTML Beispiel CL

```
<div>
  <a href=".../seite1.html" rel="first">
  <a href=".../seite4.html" rel="next">
  <a href=".../seite2.html" rel="prev">
</div>
```

Dieses HTML-Beispiel zeigt die Anreicherung der Links mit semantischen Informationen. Ein Client der die verwendeten Relationsattribute kennt, kann die Sammlung von Verweisen durchsuchen und anschließend ohne seine eigene Position zu kennen auf die nächste, vorherige oder erste Seite navigieren.

2.3.10 Design-Elemente

Zusätzlich zu den 9 H-Faktoren definiert Amundsen vier Design-Elemente. Diese Design-Elemente stellen vor allem vier charakteristische Eigenschaften von Hypermedia-Formaten dar. Aus diesem Grund werden sie im Folgenden beschrieben und im weiteren Verlauf zum Vergleich unterschiedlicher Formate verwendet.
Die vier Hypermedia-Design-Elemente sind:[22]

- Base Format
- State Transfer
- Domain Style
- Application Flow

Jeder Hypermedia-Typ basiert auf einem bestimmten zugrundeliegenden Format. Bekannte Formate sind XML, JSON, CSV und viele weitere. Ein Hypermedia-Typ kann auf einem beliebigen Basisformat aufbauen. Erst die Anreicherung des Basisformats mit zusätzlichen Metadaten und der Unterstützung von Hypermedia-Faktoren in einem bestimmten Umfang definieren einen Medientyp. Der Medientyp bestimmt außerdem, ob eine Zustandsänderung der Applikation unterstützt wird oder nicht. Man kann hier unterscheiden zwischen der Art und Weise wie eine Zustandsänderung durchgeführt werden kann. Hierbei gibt es drei Ausprägungen der Unterstützung. Keine

[21] Vgl. Amundsen, M. (2012), S. 19
[22] Vgl. Amundsen, M. (2012), S. 20

Unterstützung bei nur lesenden Formaten, Definierte Unterstützung nach einer externen Dokumentation und Ad-Hoc Unterstützung mittels einer in die Nachricht eingebetteten Hypermedia-Steuerung. Der Domain-Style beschreibt wie stark ein Medientyp an eine Problemdomäne angepasst ist. Diese Stile sind kategorisiert als spezifisch, allgemein und agnostisch. Als letztes Design Element gilt es noch den Application-Flow zu erwähnen. Hierbei geht es darum ob und in welchem Maße der Client in der Lage ist den Applikationsfluss zu steuern. Auch hier unterscheidet man drei Ausprägungen. Der Applikationsfluss kann entweder nicht, wesentlich oder hauptsächlich durch den Medientyp gesteuert werden.[23]

2.3.11 Hypermedia-Ebenen

Zusätzlich zu den Hypermedia-Faktoren und den Design-Elementen definiert Amundsen noch vier Ebenen, auf denen sich ein Hypermedia-Typ befinden kann. Diese vier Ebenen sind aus der Sicht des Clients beschrieben und unterscheiden sich nach ihren Möglichkeiten, in wie fern sie im Falle einer Veränderung angepasst werden müssen. Sie sind also gekennzeichnet durch einen bestimmten statischen und einen variablen Anteil. Statisch bezieht sich in diesem Kontext darauf, dass die statischen Anteile bei einer Veränderung oder Erweiterung der Applikation angepasst werden müssen. Die variablen Anteile hingegen können verändert werden, ohne Probleme bei der Kommunikation zwischen Client und Server zu verursachen. Entweder dem statischen oder variablen Anteil zugeordnet werden die vier Eigenschaften Content, Address, Read Write Semantics und Appflow. Diese Eigenschaften sind wie folgt definiert:[24]

- Content beschreibt die Fähigkeit Elemente einer Nachricht hinzuzufügen oder zu entfernen.
- Address ist definiert als die Möglichkeit, einen Unified Ressource Identidifier zu ändern, an den der Client seine Anfragen sendet.
- Read Write Semantics ist eine Eigenschaft, bezüglich des Schreib-Leseverhaltens. Ein Hypermedia-Typ, der diese Eigenschaft besitzt, ist in der Lage anzuzeigen, welche Elemente einer Nachricht schreibbar sind und welche Protokollfunktion zu benutzen ist.
- Appflow bezeichnet die Möglichkeit den Applikationsfluss kontextbezogen zu steuern. Der Client benötigt nur die Antwort des Servers und kann die Steuerelemente der Nachricht benutzen um sich im Applikationsfluss zu bewegen.

Als Ebene 0 definiert Amundsen serialisierte Objekte. Diese Ebene trifft auf Techniken wie SOAP und RPC-XML zu. Die Kommunikation zwischen Client und Server läuft über ein spezielles, anwendungsbezogenes Nachrichtenformat. Die Definition der Kommunikationsregeln, also welches

[23] Vgl. Amundsen, M. (2012), S. 20
[24] Vgl. Amundsen, M. (2010), `http://amundsen.com/hypermedia/scraps/#hypermedia-levels`

Format mit welchem Inhalt wo benutzt wird, muss durch eine externe Dokumentation erfolgen. Diese Regeln werden dann im Client codiert und sie verfügen über keinerlei variablen Anteil. Die Konsequenz ist, dass der Client im Falle einer Änderung in allen vier Eigenschaften angepasst werden muss.

Ebene 1 ist beschrieben als standardisierte Datenformate. Beispielhafte Medientypen sind Comma Seperated Value (CSV) oder Extendible Markup Language (XML). Die Kommunikation zwischen Client und Server läuft über ein standardisiertes Datenformat. Dies hat den Vorteil, dass beide Seiten in der Lage sind den Nachrichten zu validieren. So können Elemente oder Attribute einer Nachricht hinzugefügt oder entfernt werden und die Kommunikation läuft weiterhin fehlerfrei. Der variable Anteil beschränkt sich jedoch auf die Eigenschaft Content. Die weiteren drei Eigenschaften sind nach wie vor statisch und eine Änderung würde zu Fehlern in der Kommunikation führen.[25]

Ebene 2 erweitert die standardisierten Datenformate mit Links. Atom und AtomPub sind Medientypen dieser Ebene. Der wesentliche Unterschied ist, dass ab dieser Ebene URIs in den Nachrichten enthalten sind. URIs werden somit nicht mehr im Client codiert, sondern der Client bekommt mit jeder Nachricht URIs geliefert, denen er folgen kann. Dieser Fakt erweitert auch den variablen Anteil aus Ebene 1 um die Eigenschaft Address. Eine Anwendung, die einen Medientyp dieser Ebene verwendet, kann ein variables URI-Schema benutzen. Sollte der Server seine URIs ändern, kann er dies tun ohne die Kommunikation zu bestehenden Clients zu gefährden. Die Eigenschaften Read-Write-Semantics und Appflow sind weiter statisch.[26]

Ebene 3 ist genauso definiert wie Ebene 2, jedoch erweitert um die Möglichkeit der Applikationssteuerung. Die Kopplung zwischen Client und Server, welche einen Medientyp dieser Ebene verwenden, besteht aus einer Vereinbarung über den verwendeten Datentyp und den Elementen einer Nachricht, die zur Applikationssteuerung benutzt werden. Der Client muss in der Lage sein, die Steuerelemente einer Nachricht zu erkennen und dem Benutzer so die derzeit aktuellen Steueroptionen anzuzeigen. Hypermedia-Formate dieser Ebene sind unter anderem die Hypertext Markup Language (HTML) und VoiceXML.[27]

Betrachtet man die H-Level nach Amundsen etwas abstrakter, sind die Ebenen nach dem Grad der Kopplung zwischen Client und Server definiert. Wobei der Grad an Kopplung als Wissen verstanden werden kann, welches der Client über die Nachrichten haben muss. Dieses Wissen wird auch als 'Out of Band Knowledge' bezeichnet. Der Verzicht, dieses Wissen innerhalb des Clients

[25] Vgl. Amundsen, M. (2010), http://amundsen.com/hypermedia/scraps/#hypermedia-levels
[26] Vgl. Amundsen, M. (2010), http://amundsen.com/hypermedia/scraps/#hypermedia-levels
[27] Vgl. Amundsen, M. (2010), http://amundsen.com/hypermedia/scraps/#hypermedia-levels

zu codieren, sorgt für eine bessere Flexibilität und Wartbarkeit des Gesamtsystems. Hypermedia-Formate ab der dritten Ebene betten ihre Nachrichten in einen semantischen Kontext. Die Kopplung besteht hier dann ausschließlich aus den definierten Steuerelementen und der Einigung über das Datenformat.

3 Vergleich der Formate

3.1 Fallbeispiel Seminarverwaltung

Um den Vergleich der einzelnen Formate im Folgenden anschaulicher zu machen wird an dieser Stelle ein Fallbeispiel eingeführt. Als Beispiel dient ein System zur Seminarverwaltung, wie es bei einem Dienstleistungsunternehmen für Weiterbildung zum Einsatz kommen könnte. Angenommen wird, dass die Seminarverwaltung als typisches Backend-System eingesetzt wird. Es umfasst Schnittstellen zu anderen unternehmensinternen Anwendungen und zu einer Webseite, die es externen Kunden ermöglicht Seminare zu buchen. Die Seminarverwaltung hat primär die Aufgabe Seminaranfragen entgegen zu nehmen und den Lebenszyklus einer Anfrage zu verwalten. Die vollständige Modellierung einer solchen Anwendung würde den Rahmen dieser Betrachtung übersteigen. Aus diesem Grund sei vereinfacht angenommen, dass die Seminarverwaltung nur für folgende Aufgaben verantwortlich ist:

- Erfassung und Bearbeitung von Seminaranfragen mit Datum, Preis, Seminarthema und Rechnungsanschrift des Kunden
- Verwaltung der Seminaranfragen mit den Zuständen Anfrage eingegangen, in Bearbeitung, Termin bestätigt, nicht durchführbar, storniert und durchgeführt.

Aus den genannten Aufgaben ergibt sich der in Abbildung 7 dargestellte Kontext für die Seminarverwaltung. Kundenanfragen werden ausschließlich in der Seminarverwaltung bearbeitet und resultieren entweder aus einer telefonischen oder einer Online-Anfrage. Daten über die angebotenen Seminare sowie die Kundendaten werden in separaten unternehmensinternen Anwendungen gehalten.

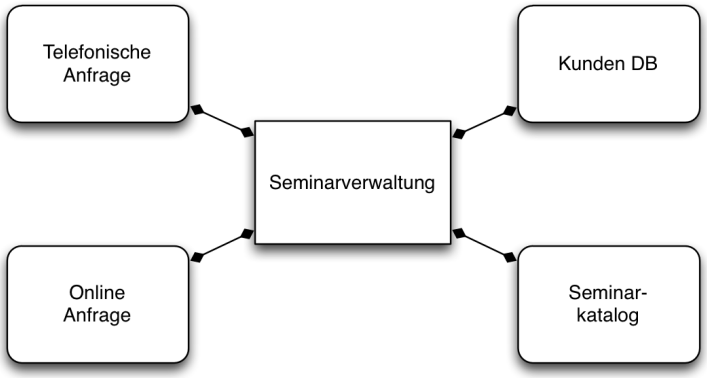

Abbildung 7: Kontextdiagramm Seminarverwaltung

Um konkrete Nachrichten im späteren Verlauf des Vergleichs darstellen zu können werden die in Tabelle 1 beschriebenen Ressourcen der Seminarverwaltung und ihre jeweils zulässigen Methoden zu Grunde gelegt. Die zulässigen Methoden legen fest, welche Arten von Anfragen auf die Ressource erlaubt sind. Die HTTP-Verben werden stets entsprechend ihrer Spezifikation nach RFC2616 der IETF verwendet. Die Nachrichten sind dann als Repräsentationsformen der Ressourcen definiert. Eine GET-Anfrage auf '/seminare/123' liefert als Antwort eine Repräsentation der Seminaranfrage mit der ID 123. Eine PUT- oder POST-Anfrage auf die selbe Ressource, aktualisiert diese wenn sie bereits besteht. Sollte die Ressource noch nicht existieren, wird eine neue Ressource angelegt. Anfragen auf andere Ressourcen verhalten sich analog zu dieser Vorgehensweise.

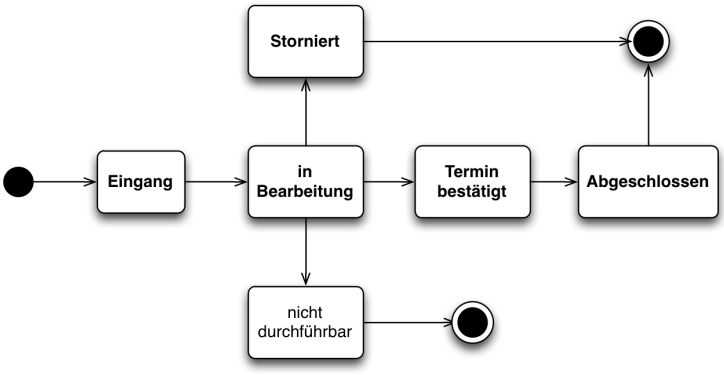

Abbildung 8: Zustandsdiagramm Seminarverwaltung

Abbildung 8 zeigt den der Seminarverwaltung zu Grunde liegenden Lebenszyklus einer Kundenanfrage. Eine Anfrage durchläuft während ihrer Lebensdauer die verschiedenen Bearbeitungsstufen und endet je nach Ausgang der Bearbeitung. Der Ablauf sieht vor, dass nach Eingang einer Anfrage der Zustand auf 'in Bearbeitung' geändert und die Durchführbarkeit geprüft wird. Solange

sich die Anfrage in Bearbeitung befindet, hat der Kunde die Möglichkeit seine Anfrage zu stornieren. Sollte dem Kundenwunsch nicht entsprochen werden können, ändert sich der Status auf 'nicht durchführbar'. Wenn alle Prüfungen der Anfrage erfolgreich verlaufen sind, wird ein Termin festgelegt und der Status ändert sich in 'Termin bestätigt'. Von nun an ist keine Stornierung mehr möglich. Nach der Durchführung des Seminars wird der Status auf 'abgeschlossen' gesetzt und der Lebenszyklus endet. Zustände einer Anfrage können während der Lebensdauer durch die statusspezifischen Ressourcen abgefragt werden.

Tabelle 1: Ressourcen der Seminarverwaltung

Ressource	**URI**	**Methode**
Liste aller Seminaranfragen	/seminare/	GET, POST
Seminaranfrage	/seminare/(id)	GET, PUT, POST
Seminaranfragen im Eingang	/seminare/?status=eingang/	GET
Seminaranfrage in Bearbeitung	/seminare/?status=bearbeitung/	GET
Seminaranfrage Termin bestätigt	/seminare/?status=bestaetigt/	GET
Abgeschlossene Seminaranfragen	/seminare/?status=erledigt/	GET
Seminaranfrage nicht durchführbar	/seminare/?status=nd/	GET
Seminaranfrage storniert	/seminare/(id)/storno/	GET
Liste stornierter Seminaranfragen	/storno/	GET, POST
Stornierung	/storno/(id)	GET

Die Repräsentation einer Bestellung enthält folgende Elemente:

- Eingangs- und Durchführungsdatum
- Preis des Seminars
- Status der Anfrage
- Seminarthema
- Kunde

Zusätzlich wird die Repräsentation einer Stornierung definiert mit folgenden Inhalten:

- Kunde
- Seminar
- Preis
- Stornierungsdatum

Der Einsatz dieses Fallbeispiels wird im nachstehenden Vergleich als Annahme getroffen, um den Einsatz verschiedener Hypermedia-Formate als Repräsentation der Ressourcen zu zeigen. Nachrichten, als Auszüge der Kommunikation dieses Systems, sollen die Unterschiede zwischen den zum Tragen kommenden Formaten darstellen.

3.2 Vergleichsgrundlage

Innerhalb dieser Arbeit werden die Hypermedia-Faktoren, die Hypermedia-Ebenen und die vier Design-Elemente als Grundlage verwendet, um verschiedene Formate untereinander vergleichbar zu machen. Diese Ansicht ist nur als Vergleich, nicht jedoch als Bewertung zu sehen, da Hypermedia Typen nicht zwangsläufig alle H-Faktoren und Design-Elemente unterstützen müssen. Die Ausprägung der Unterstützung eines Formats bezüglich der Faktoren und Ebenen sollte stets auf einen konkreten Anwendungsfall bezogen sein. So kann ein rein zum Lesen ausgelegter Medientyp z.B. auf das Design-Element des Applikationsflusses verzichten, weil es im Kontext seiner Anwendung nicht erforderlich ist. Ein gängiges Beispiel hierfür ist ein Atom-Feed-Reader. Der Medientyp beinhaltet nur eine kurze Vorschau und einen Link auf den gesamten Artikel. Funktionen, die den Applikationsfluss steuern sind in diesem Fall nicht notwendig.
Der Vergleich verschiedener Formate wird auf Grundlage der in Tabelle 2 dargestellten Merkmale und deren Ausprägungen durchgeführt. Die Spalte der H-Faktoren stellt alle Faktoren dar, die durch das Format zum Einsatz gebracht werden können. Das Hypermedia-Level kann Werte von Null bis Drei annehmen und sagt etwas über die entstehende Kopplung zwischen den Teilnehmern aus. Die Spalte kann auch interpretiert werden als ein Grad des benötigten 'Out-of-Band Knowledge'. Wobei zu Grunde gelegt wird, dass wie im Kapitel 2.3.11 beschrieben, auf den höheren Ebenen weniger Wissen benötigt wird und eine geringere Kopplung entsteht. Die Design Elemente illustrieren die allgemeinen Eigenschaften eines Formats und beziehen sich in ihren Ausprägungen auf die in Kapitel 2.3.10 beschriebenen Merkmale.

Neben dem mittels Tabelle 2 durchgeführten Vergleich werden exemplarisch Nachrichten bezogen, auf das Anwendungsbeispiel dargestellt und die gegebenenfalls bereits definierten Relationsattribute vorgestellt. Des weiteren werden innerhalb dieses Vergleichs nur die in der Spezifikation des Formats beschriebenen Eigenschaften betrachtet.

Tabelle 2: Vergleichsgrundlage

H-Faktoren	H-Level	Design Elemente
LE, LO, LT, LI, LN, CR, CU, CM, CL	Level	Base Format, State Transfer, Domain Style, Application Flow

3.3 JSON und XML

JSON wurde ursprünglich abgeleitet aus dem ECMAScript-Standard, aus dem auch JavaScript stammt und definiert nur eine kleine Regelmenge bezüglich seiner Formatierung und gültiger Elemente. Das Ziel bei der Erstellung von JSON war es, ein leichtgewichtiges und sprachunabhängiges Datenaustauschformat zu entwerfen. JSON selbst versteht sich als eine serialisierte Repräsentation von strukturierten Daten. Strukturierte Daten bezieht sich in diesem Kontext auf den im Rahmen von JavaScript verwendeten Objektbegriff. JSON kennt im wesentlichen zwei Arten von Repräsentationen. Es wird unterschieden zwischen primitiven und strukturierten Datentypen. Zu den primitiven Datentypen zählt JSON Strings, numerische Werte, boolsche Werte und null. Für die Repräsentation von strukturierten Datentypen stehen Arrays und Objekte zur Verfügung. Ein Array ist definiert als eine geordnete Folge von Werten und ein Objekt als eine ungeordnete Folge von Schlüssel/Wert-Paaren.[28]
Die Repräsentation einer Seminaranfrage und einer entsprechenden Stornierung auf Grundlage des Fallbeispiels zeigen die Listings 10 und 11.

Listing 10: Seminaranfrage in JSON

```
{
    "seminaranfrage":        {
        "anfrage_id": "31235",
        "eingangsdatum": "2013-05-05",
        "status": "storniert",
        "stornierung_id": "5512",
        "preis": "1000",
        "thema": "Hypermedia",
        "seminar": "http://seminarkatalog.de/seminare/123",
        "kunde_name": "Muster GmbH",
        "kunde_addr": "Musterstr. 5b",
        "kunde": "http://kundendb.de/kunden/567312"
                              }
}
```

[28] Vgl. Crockford, D. (2006), http://www.ietf.org/rfc/rfc4627

Listing 11: Stornierung in JSON

```
{
    "stornierung": {
        "stornierung_id": "5512",
        "stornierungsdatum": "2013-06-01",
        "anfrage_id": "31235",
        "eingangsdatum": "2013-05-05",
        "preis": "1000",
        "thema": "Hypermedia",
        "seminar": "http://seminarkatalog.de/seminare/123",
        "kunde_name": "Muster GmbH",
        "kunde_addr": "Musterstr. 5b",
        "kunde": "http://kundendb.de/kunden/567312"
                    }
}
```

Die JavaScript Object Notation wird teilweise von den im Vergleich folgenden Medientypen als Basisformat genutzt. Eine gesonderte Betrachtung ist notwendig, weil es selbst keine Möglichkeit bietet semantische Informationen hinzuzufügen. Innerhalb dieser Arbeit wird reines JSON ausschließlich als Basis für andere Formate betrachtet. JSON versteht sich selbst als ein reines Datenaustauschformat ohne Möglichkeit, die transportierten Daten genauer zu beschreiben. Abgesehen davon bietet es aufgrund der kleinen Regelmenge, eine sehr leichtgewichtige Grundlage wenn es als Basisformat verwendet und mit Metadaten angereichert wird.
Neben JSON wird oftmals auch die von der W3C spezifizierte Extensible Markup Language (XML) verwendet. XML selbst ist als Sprache umfangreicher definiert als JSON, bietet jedoch ebenfalls per Spezifikation keine semantische Anreicherung der Dokumente an. Von einer XML-Instanz wird stets gefordert, dass sie wohlgeformt ist. Es ist also eine Bedingung, jedes Element nach dem Öffnen auch wieder zu schließen. Grundsätzlich unterscheidet XML zwischen Elementen und Attributen. Attribute können in einer XML-Instanz nur innerhalb eines Elementes erstellt werden und werden in der Regel dazu genutzt um ein Element genauer zu beschreiben. Listing 12 zeigt eine Seminaranfrage als ein wohlgeformtes XML-Dokument.

Listing 12: Stornierung in XML

```
<?xml version="1.0" encoding="UTF-8">
<seminaranfragen>
        <seminaranfrage>
                <anfrage_id>31235</anfrage_id>
                <eingangsdatum>2013-05-05</eingangsdatum>
                 <status>storniert</status>
                <stornierung_id>5512</stornierung_id>
                <preis>1000</preis>
```

```
            <thema>Hypermedia</thema>
            <seminar>http://seminarkatalog.de/seminare/123</seminar>
            <kunde_name>Muster GmbH</kunde_name>
            <kunde_addr>Musterstr. 5b</kunde_addr>
            <kunde>http://kundendb.de/kunden/567312</kunde>
        </seminaranfrage>
</seminaranfragen>
```

Im Gegensatz zu einem JSON-Dokument, wirkt das äquivalente XML-Dokument überladen. Diese Tatsache ist der Anforderung geschuldet, dass ein XML-Dokument wohlgeformt sein muss. Dieser zusätzliche Inhalt sorgt jedoch für eine leichte Validierung der Instanz. Zusätzlich unterstützt XML die Verwendung von Namensräumen und die Validierung durch entsprechende Schemata. XML ist eine weit verbreitete Sprache und es existieren Parser und Validatoren für sehr viele Programmiersprachen. So kann XML leicht als ein Basisformat für verschiedene Medientypen eingesetzt werden. Alle Medientypen dieses Vergleichs verwenden entweder JSON oder XML als Basisformat und fügen ihm jeweils typspezifische Regularien hinzu. Manche Medientypen bieten auch eine Repräsentation in beiden Formaten an.

3.4 Collection + JSON

Collection + JSON wurde von Mike Amundsen als Medientyp entwickelt und bei der IANA registriert. Nach Spezifikation ist Collection + JSON die Definition des Formats sowie der dazugehörigen semantischen Regeln. Dieser Medientyp bietet Schreib- und Lesemöglichkeiten, sowie die Verwaltung und Abfrage von einfachen Datensammlungen. Neben dem definierten Aufbau einer Collection-Repräsentation bietet Collection + JSON die Möglichkeit vorgefertigte Abfragen, sowie Vorlagen zum Aktualisieren und Anlegen einer Ressource zu verwenden. Abfragen und Vorlagen werden stets vom Server als ein Teil der Antwort zum Client gesendet. Der Client benötigt daher kein Wissen über die Art und Weise wie nach einer Ressource gesucht wird oder eine neue Ressource angelegt wird. Eine gültige Collection + JSON Nachricht muss mindestens, wie in Listing 13 dargestellt, ein Collection-Objekt enthalten.[29]

Listing 13: Minimale Repräsentation Collection + JSON

```
{
    "collection":     {
        "version": "1.0",
        "href": "http://seminarverwaltung.de"
                    }
}
```

[29] Vgl. Amundsen, M. (2013), http://amundsen.com/media-types/collection/format/

Eine Collection wiederum kann mehrere items beinhalten, wobei jedem item mittels href-Attribut eine URI zugewiesen wird. Zusätzlich können zwei Arraystrukturen, data und links, einem item hinzugefügt werden. Um die Elemente dieser Arrays mit zusätzlichen semantischen Informationen anzureichern, kann die name-Eigenschaft innerhalb der Strukturen verwendet werden. Collection + JSON stellt im Fehlerfall dem Client ein error-Object mit den Eigenschaften title, code und message zur Verfügung.[30]
Zur Verdeutlichung der Möglichkeiten von Collection + JSON wird im Folgenden der Einsatz dieses Medientyps in der Seminarverwaltung dargestellt. Listing 14 zeigt die Antwort auf eine initiale Anfrage an den Server der Seminarverwaltung.

Listing 14: Initiale Anfrage Collection + JSON

```
{
    "collection":{
        "version": "1.0",
        "href": "http://seminarverwaltung.de",
        "links": [
                {
                "rel": "seminare",
                "href": "http://seminarverwaltung.de/seminare"
                },
                {
                "rel": "stornierungen",
                "href": "http://seminarverwaltung.de/storno"
                }
                ]
        }
}
```

Als Antwort auf die initiale Anfrage erhält der Client eine Collection, die zwei Verweise enthält, denen er folgen kann. Diese Verweise bestehen ihrerseits jeweils aus einem rel- und einem href-Wert. Der rel-Wert beschreibt die Art der Ressource auf die der href-Wert referenziert. Durch die Möglichkeit, Verweise mit Beziehungswerten zu dekorieren, kann die Nachricht mit semantischen Informationen angereichert werden. Aufgrund dieser Nachricht kann der Client entscheiden ob er als nächstes entweder eine Repräsentation der Seminaranfragen oder Stornierungen erhalten möchte. Zur Interpretation der Nachricht benötigt der Client also nur Wissen über die Bedeutung der Relationswerte. Die tatsächlichen Adressen der Ressourcen bleiben für den Client komplett transparent und eine Kopplung entsteht nur auf Basis der Relationswerte. Listing 15 zeigt die Repräsentation der Seminaranfragen, für den Fall das der Client dem Relationswert 'seminare' gefolgt ist.

[30] Vgl. Amundsen, M. (2013), http://amundsen.com/media-types/collection/format/

Listing 15: Seminaranfrage Collection + JSON

```
{"collection": {
 "version": "1.0",
 "href": "http://seminarverwaltung.de",
 "links": [{
    "rel": "stornierungen",
    "href": "http://seminarverwaltung.de/storno"
      }],

"items": [
  {"href": "http://seminarverwaltung.de/seminare/31235",
   "data": [
    {"name": "anfrage_id", "value": "31235", "prompt": "Anfrage ID" },
    {"name": "eingangsdatum", "value": "2013-05-05", "prompt": "Eingang"},
    {"name": "status", "value": "storniert", "prompt": "Status"},
    {"name": "preis", "value": "1000", "prompt": "Preis"},
    {"name": "thema", "value": "Hypermedia", "prompt": "Thema"}],

 "links": [
    {"rel": "seminarbeschreibung",
     "href": "http://seminarkatalog.de/seminare/123", "prompt": "Seminar"},
    {"rel": "kundenbeschreibung",
     "href": "http://kundendb.de/kunden/567312", "prompt": "Kunde"},
    {"rel": "stornierung",
     "href": "http://seminarverwaltung.de/storno/5512", "prompt": "Stornierung"}]}],

 "queries": [{
     "rel": "filter",
     "href": "http://seminarverwaltung.de/seminare/",
     "prompt": "Filter",
     "data": [{
      "name": "status",
      "value": ""}]}],

 "template": {"data": [
    {"name": "eingangsdatum", "value": "", "prompt": "Eingang"},
    {"name": "status", "value": "", "prompt": "Status"},
    {"name": "preis", "value": "", "prompt": "Preis"},
    {"name": "thema", "value": "", "prompt": "Thema"}]}}}
```

Auf Grundlage dieser Nachricht kann der Client die weitere Interaktion mit der Anwendung steuern. Voraussetzung dafür ist, dass der Client die verwendeten rel- und name-Attribute interpretieren kann. Die links-Arrays ermöglichen dem Client die Navigation durch die Anwendung. Von dieser Stelle aus kann er sich zu den Stornierungen, der Ansicht des Seminars und den verknüpften Beschreibungsdatensätzen, sowie der verknüpften Stornierung bewegen. Das Erstellen einer Seminaranfrage kann der Client anhand des template-Arrays durchführen. Diese Vorlage definiert, welche Werte mindestens gegeben sein müssen, um eine neue Ressource zu erzeugen. Durch diese Eigenschaft ermöglicht Collection + JSON die Erstellung einer neuen Ressource auf eine gewisse Art und Weise zu reglementieren. Die korrekte Erstellung einer Ressource liegt nicht in der Verantwortung des Client allein. Vielmehr ist dieses Template eine Art Vereinbarung zwischen Client und Server. Der Dienstanbieter verspricht mit dieser Vorlage, sich um das Anlegen einer gültigen Ressource zu kümmern, wenn er vom Dienstkonsumenten die geforderten Werte übergeben bekommt. Um eine Ressource zu löschen oder zu aktualisieren, wird das href-Attribut des items verwendet. Ein Löschen wäre in diesem Beispiel mit einer DELETE-Anfrage auf die gewünschte Ressource möglich. Der Server sollte diese Anfrage entsprechend mit einem ’204 No Content’ quittieren. Des Weiteren wird dem Client mit dem queries-Array eine gefilterte Ansicht zur Verfügung gestellt. Wird an dieser Stelle der String ’bestaetigt’ übergeben, kann der Client die URI `"http://seminarverwaltung.de/seminare/?status=bestaetigt"` generieren und an den Server senden. Entsprechend der Definition des Fallbeispiels bekommt er auf dieser URI eine Liste aller Seminaranfragen mit dem Status bestätigt. Listing 16 zeigt die Repräsentation eines Fehlers falls eine Interaktion nicht durchgeführt werden konnte.

Listing 16: Fehlerobjekt Collection + JSON

```
{"error": {
  "title": "Not found",
  "code": "404",
  "message": "Ressource nicht gefunden"
           }}
```

Collection + JSON hat alle Link-Faktoren umgesetzt. Embeddet- und Outbound-Links sind durch das href-Attribut realisiert. Die dadurch identifizierten Ressourcen können mit idempotenten und nicht idempotenten Operationen angesprochen werden und sind über entsprechende Relationswerte semantisch beschrieben. Das Queries-Array entspricht dem H-Faktor der Templated-Links. Diese vorgefertigten Abfragen geben dem Client die Möglichkeit, definierte Strings zu übergeben und eine entsprechende URI zu konstruieren. Durch die Option Elemente mit rel- und name-Attributen zu dekorieren ist der H-Faktor Link-Annotation-Controls erfüllt und die Nachrichten können mit semantischen Informationen angereichert werden. Diese semantischen Informationen sagen dem Client etwas über die Natur der referenzierten Ressource und er kann sich entsprechend seiner Kenntnis verhalten. Das Listing 17 zeigt nochmals das queries-Array als einen Ausschnitt der Nachricht und die Umsetzung der Link-Faktoren.

Listing 17: H-Faktoren Collection + JSON

```
"queries": [{                                          <<LT
 "rel": "filter",                                      <<CL
 "href": "http://seminarverwaltung.de/seminare/",      <<LE, LO, LI, LN
 "prompt": "Filter",
 "data": [{
   "name": "status",
   "value": ""}]}]
```

Die H-Faktoren Read-, Update- und Method-Controls werden nicht unterstützt, weil es nicht möglich ist, Operationen auf bestimmte Formate oder Methoden einzuschränken. Sollte eine Seminaranfrage derart erweitert werden, dass sie das Firmenlogo des entsprechenden Kunden beinhalten soll, kann eine Formatbeschränkung dieser Ressource sinnvoll sein. Wenn die Seminarverwaltung nur mit JPEG-Dateien arbeiten kann, sollte die Ressource auch auf genau diesen Typ beschränkt werden. Eine solche Einschränkung lässt sich mit Collection + JSON nicht abbilden. Analog dazu verhalten sich die Methoden, wie eine Ressource angesprochen werden sollte. Um Idempotenz zu gewährleisten könnte die Anforderung entstehen, dass alle Ressourcen, die beschrieben werden dürfen, nur über PUT anzusprechen sind. POST wird dann zu einer nicht zugelassenen Methode. Collection + JSON bietet keine Möglichkeit dieses Regelwerk innerhalb einer Nachricht darzustellen. Weitere Einschränkungen gibt es bei der Komplexität der repräsentierten Datenstrukturen. Collection + JSON eignet sich gut um einfache Sammlungen und Objekte darzustellen. Es ist jedoch nicht möglich, ineinander verschachtelte Array-Strukturen zu erstellen. Eine Zustandsänderung der Anwendung wird Ad-Hoc durch eingebettete Verweise und Relationsattribute durchgeführt. Die Steuerung des Applikationsflusses kann somit hauptsächlich durch den Medientyp erfolgen.
Abschließend muss noch erwähnt werden, dass Collection + JSON ein sehr allgemeiner Medientyp ist. Er ist selbst nicht angepasst an eine spezifische Problemdomäne. Eine problembezogene Anpassung kann jedoch durch den Einsatz der Relationsattribute umgesetzt werden. Verwendete Relationsbegriffe müssen dann in einer externen Dokumentation erfasst und beschrieben werden. Diese Beschreibung sollte zum einen die Semantik der Beziehung und zum anderen die möglichen Werte und Ausprägungen enthalten. Auf dieser Grundlage kann ein Dienstkonsument die Verweise interpretieren und hat eine Vorstellung von der übermittelten Repräsentation. Collection + JSON kann problemlos in Umgebungen eingesetzt werden, die hauptsächlich aus vergleichsweise einfachen Daten- und Objektstrukturen bestehen. Die Steuerung der Anwendung sowie das Einbetten von semantischen Informationen kann allein durch einen Nachrichtenaustausch realisiert werden. In dieser Umgebung muss allerdings auf die Möglichkeit verzichtet werden, die Kommunikation strikt zu reglementieren. Diese Einschränkung beruht auf der fehlenden Unterstützung der H-Faktoren CR, CU und CM. Tabelle 3 zeigt die Eigenschaften von Collection + JSON im Überblick.

Tabelle 3: Überblick Collection + JSON

H-Faktoren	H-Level	Design Elemente
LE, LO, LT, LI, LN, CL	Level 3	Base Format = JSON, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich

3.5 JSON-LD

JSON for Linking Data (JSON-LD) wurde entwickelt um in einer leichtgewichtigen Nachricht verknüpfte Daten darzustellen. Das Ziel von JSON-LD ist es, einem Dienstkonsumenten eine Nachricht zu senden und der Konsument kann anhand dieser Nachricht auf andere verknüpfte Daten und Repräsentationen zugreifen. Dieses Vorgehen soll mittels in der Nachricht eingebetteter Verweise erfolgen. JSON-LD soll dabei vollständig JSON kompatibel bleiben, sodass bereits vorhandene Parser und Anwendungen verwendet und erweitert werden können. Die Transformation von JSON zu JSON-LD sollte im Idealfall ohne Änderung des bestehenden JSON-Dokuments durchführbar sein. JSON-LD verwendet grundsätzlich die gleichen formalen Definitionen wie JSON. Es gibt Objekte, Arrays, Strings, numerische Werte, boolsche Werte und null. Daten werden bei JSON-LD in Form eines gerichteten Graphen modelliert, wobei ein Knoten entweder auf einen anderen Knoten oder auf Werte zeigen kann. Ein Knoten selbst wird hierbei stets durch eine URI identifiziert. Die Kanten des Graphen sollten, sofern dies möglich ist, immer mit einer URI beschriftet werden. Innerhalb eines JSON-LD Dokuments werden die Beschriftungen der Kanten als Eigenschaften erfasst. Diese Eigenschaften sagen etwas über den semantischen Kontext des folgenden Wertes oder des folgenden Knotens aus. Der semantische Kontext selbst wird in JSON-LD durch das Kontext-Konzept realisiert. Innerhalb des Kontext wird festgelegt, welche semantische Bedeutung ein Knoten oder ein Wert besitzt. Vereinfacht gesagt dient der Kontext dazu, die verwendeten Begriffe auf URIs abzubilden. Die Begriffsdefinition kann in JSON-LD entweder aus einer URI oder einem JSON-Objekt bestehen. Das Verwenden eines Objekts zur Definition wird erweiterte Begriffsdefinition genannt. Auf diese Weise kann ein Begriff genauer spezifiziert werden. Zusätzlich zu der URI, die auf den semantischen Kontext verweist, können Typen- und Sprachinformationen mit dem Begriff assoziiert werden. Die zwei elementaren Schlüsselwörter für die Assoziation sind @id und @type. @id legt die Identität eines Knotens fest und entspricht einer URI. Das @type-Schlüsselwort legt fest, um was für eine Art von Ressource es sich hierbei handelt. Diese beiden Schlüsselwörter sorgen somit für die Identifizierung einer Ressource und für eine Anreicherung der Nachricht mit semantischen Informationen.[31]
Die Listings 18 und 19 zeigen eine Kontext-Datei und eine Instanz, die diesen Kontext verwendet. In diesem Beispiel ist der Kontext in einer separaten Datei deklariert. JSON-LD unterstützt auch eine In-Line Deklaration des Kontext.

[31] Vgl. Sporny, M. u.a. (2013), `http://json-ld.org/spec/latest/json-ld/`

Listing 18: Kontext JSON-LD

```
{
"@context" : {
"name":        "http://semantik.de/person/name",
"alter":
{
"@id":         "http://semantik.de/person/alter",
"@type":       "http://dataytypes.com/integer"
}
"avatar":      {
"@id":         "http://semantik.de/person/bild",
"@type":       "@id",
}}}
```

Listing 19: Instanz JSON-LD

```
{
"@context":    "http://meine-semantik.de/person.jsonld",
"@id":         "http://mustermann.de/max",
"name":        "Max Mustermann",
"alter":       "30",
"avatar":      "http://mustermann.de/max/pic/max.jpeg"
}
```

Wie an diesem Beispiel zu sehen ist, kann sich der Kontext aus unterschiedlichen internen und externen Quellen zusammensetzen. Dieses Beispiel würde den in Tabelle 4 dargestellten Datensatz erzeugen.

Tabelle 4: Datensatz JSON-LD

Knoten ID	Eigenschaft	Wert	Typ des Werts
http://mustermann.de /max	http://semantik.de/ person/name	Max Mustermann	
http://mustermann.de /max	http://semantik.de /person/alter	30	http://dataytypes.com /integer
http://mustermann.de /max	http://semantik.de /person/bild	http://mustermann.de /max/pic/max.jpeg	URI

Strings, die mit einem @id verknüpft sind, werden stets als URIs behandelt. Die Kombination '@type': '@id' besagt, dass der Wert immer als URI gesehen werden kann. JSON-LD kann mit relativen und absoluten URIs umgehen. Die Spezifikation der Datentypen bietet noch weitere Möglichkeiten. So kann man einem Knoten oder Wert per Array mehrere Typen zuweisen. Der Wert

für 'alter' könnte beispielsweise die Typen integer und float annehmen, wenn die Eigenschaft im Kontext entsprechend definiert ist. Arrays werden in JSON-LD unterschieden in list und set. Set ist der Standard in JSON-LD und repräsentiert eine nicht geordnete Array-Struktur. Wenn eine Ordnung der Daten gewünscht ist, kann mit dem Schlüsselwort @list ein sortiertes Array erzeugt werden. Listen von Listen sind nicht erlaubt. Anders als bei Listen ist die Einbettung eines Knoten in einen anderen Knoten gültig. Dieser wird dann als eine Eigenschaft des Eltern-Knotens geführt. JSON-LD bietet die Möglichkeit Strings mit Sprachangaben zu versehen und die Werte in unterschiedlichen Sprachen darzustellen. So kann eine Nachricht verschiedene Sprachen in ihren Repräsentationen haben und der Client verwendet die von ihm bevorzugte Sprache.[32] Listing 20 zeigt ein Beispiel für internationalisierte Strings.

Listing 20: Internationalisierte Strings

```
{
"@context":    {
...
"beschreibung": {
"@id":         "http://meine-semantik.de/beschreibung",
"@container":  "@language"}
...
}
"@id":         "http://mustermann.de/max",
"name":        "Max Mustermann",
"alter":       "30",
"avatar":      "http://mustermann.de/max/pic/max.jpeg",
"beschreibung":
{
"de":          "Mitarbeiter des Monats",
"en":          "Employee of the month"}
}
```

An dieser Stelle ist die grundlegende Betrachtung des Formats JSON-LD abgeschlossen und die Eignung des Formats für das Fallbeispiel der Seminarverwaltung wird überprüft. Um die Nachrichten semantisch anreichern zu können, muss ein Kontext für die Applikation definiert werden. Listing 21 zeigt den Kontext der Seminarverwaltung. Es sei vereinfacht angenommen, dass ein Vokabular existiert, welches alle genutzten Eigenschaften und Typen beschreibt. Dieses Vokabular ist die semantische Grundlage der Applikation und definiert Dinge der realen Welt. Die Definition beinhaltet unter anderem die Bedeutung des Status bezogen auf eine Seminaranfrage. Sollte dieses Vokabular nicht ausreichen, so ist es möglich, weitere Kontext zu definieren. Diese können dann unterschiedliche Präfixe verwenden. So kann ein und der selbe Begriff in verschiedenen Kontext verwendet werden. Der Begriff muss hierfür in der Form 'Präfix : Begriff' erstellt

[32] Vgl. Sporny, M. u.a. (2013), http://json-ld.org/spec/latest/json-ld/

werden. Da es möglich ist, einen Kontext auf allen Ebenen des Dokuments zu erstellen, gilt es außerdem zu beachten, dass ein tiefer gelegener Kontext den darüber liegenden überschreibt. Im Fall von JSON-LD repräsentiert das Vokabular die externe Dokumentation und das vom Dienstkonsumenten benötigte Wissen. An dieser Stelle wird definiert, welche Typen verwendet werden, welche Bedeutung sie im Kontext der Applikation haben und wie die gültige Repräsentation eines bestimmten Typs auszusehen hat.

Listing 21: Kontext für Seminarverwaltung

```
{
"@context":    {
"@vocab":      "http://seminarverwaltung.de/semantik"
}}
```

Listing 22 zeigt die Antwort der Seminarverwaltung auf eine initiale Anfrage.

Listing 22: Initiale Anfrage JSON-LD

```
{"@context":   "http://seminarverwaltung.de/kontext.jsonld",
"root":
[{
"@id":         "http://seminarverwaltung.de/seminare",
"@type":       "http://seminarverwaltung.de/semantik/seminare"},
{
"@id":         "http://seminarverwaltung.de/storno",
"@type":       "http://seminarverwaltung.de/semantik/stornierungen"}]}
```

Diese initiale Antwort enthält einen Verweis auf den genutzten Kontext, sowie das root-Objekt. Das Wurzel-Objekt enthält die beiden gültigen Verweise zur weiteren Navigation durch die Anwendung. Die Semantik dieser beiden Einträge ist durch das @type Schlüsselwort deklariert. Hinter der URI verbirgt sich die semantische Bedeutung der Seminare und Stornierungen. Nach Spezifikation müssen alle Strings, die auf das @id-Schlüsselwort folgen, URIs sein. Somit kann der Client den Verweisen durch den @id-Schlüssel folgen und die Bedeutung entsprechend des @type Schlüsselwortes erkennen. Listing 23 zeigt die Antwort der Seminarverwaltung für den Fall, dass der Client dem Verweis auf die Seminare gefolgt ist.

Listing 23: Seminaranfrage JSON-LD

```
{"@context":   "http://seminarverwaltung.de/kontext.jsonld",
"seminaranfragen":
[{
"@id":                "http://seminarverwaltung.de/seminare/31235",
"@type":              "http://seminarverwaltung.de/semantik/seminaranfrage",
"eingangsdatum":      "2013-05-05",
"status":             "storniert",
"preis":              "1000",
"thema":              "Hypermedia",
"seminarbeschreibung": {
"@id":                "http://seminarkatalog.de/seminare/123
"@type":              "http://seminarverwaltung.de/semantik/seminarbeschreibung},
"kundenbeschreibung": {
"@id":                "http://kundendb.de/kunden/567312
"@type":              "http://seminarverwaltung.de/semantik/kundenbeschreibung},
"stornierung": {
"@id":                "http://seminarverwaltung.de/storno",
"@type":              "http://seminarverwaltung.de/semantik/stornierungen"}}],

"filter":             {
"eingang":            "http://seminarverwaltung.de/seminare/?status=eingang",
"bearbeitung":        "http://seminarverwaltung.de/seminare/?status=bearbeitung",
"bestaetigt":         "http://seminarverwaltung.de/seminare/?status=bestaetigt",
"erledigt":           "http://seminarverwaltung.de/seminare/?status=erledigt",
"nd":                 "http://seminarverwaltung.de/seminare/?status=nd"}
}
```

Als Antwort liefert die Seminarverwaltung eine Nachricht, die ein Anfragen-Array und ein Filter-Objekt beinhaltet. Auch hier kennzeichnet das @id-Schlüsselwort immer ein URI dem der Dienstkonsument folgen kann. Die darauf folgende Antwort sollte eine Repräsentation der Ressource mit weiteren Informationen beinhalten. Informationen wie Eingangsdatum, Preis, Status und Thema bezüglich der Anfrage sind als Eigenschaften des Objekts modelliert. Zusätzlich beinhaltet eine Seminaranfrage noch die drei eingebetteten Objekte Stornierung, Seminar- und Kundenbeschreibung. Um die jeweiligen Details zu den eingebetteten Objekten abzufragen, kann der Client den Identitäten der Objekte folgen. Das Filter-Objekt stellt dem Client eine Reihe gefilterter Ansichten zur Verfügung. Damit er diesen Verweisen folgen kann, muss im Kontext der Instanz die Kombination '@type' : '@id' für alle Filterbegriffe gesetzt sein. So sind die Begriffe eingang, bearbeitung, bestätigt, erledigt und nd definiert als Identitäten und werden wie URIs behandelt.
Mit JSON-LD wäre es zudem möglich, die Seminarverwaltung multilingual zu betreiben. Ange-

nommen das Thema einer Seminaranfrage soll in mehreren Sprachen verfügbar sein. Dann muss der Kontext analog zu Listing 20 erweitert werden. Der internationalisierte Auszug einer Seminaranfrage ist in Listing 24 dargestellt. Ein Client könnte dann mit 'objekt.thema.en' die englische Darstellung des Seminarthemas abfragen.

Listing 24: Internationalisierte Seminaranfrage JSON-LD

```
{
...
"@id":                "http://seminarverwaltung.de/seminare/31235",
"@type":              "http://seminarverwaltung.de/semantik/seminaranfrage",
"eingangsdatum":      "2013-05-05",
"status":             "storniert",
"preis":              "1000",
"thema":{
       "de":          "Hypermedia macht Spass",
       "en":          "Hypermedia is fun"}
...
}
```

Durch das wesentliche Konzept von JSON-LD, Identitäten müssen immer URIs sein, erfüllt dieses Format die Hypermedia-Faktoren Embedding-, Outbound-, Idempotent- und Non-Idempotent-Links. Alle Knoten bekommen einen @id-Schlüssel, dem gefolgt werden kann und anschließend eine Repräsentation der Ressource zurück liefern. Objekte werden bei JSON-LD durch das Schlüsselwort @type mit semantischen Informationen angereichert. Diese beziehen sich meist auf einen oder mehrere Kontext, in denen die Bedeutungen definiert sind. Der Client wird so in die Lage versetzt die Objekte zu interpretieren und sich entsprechend der Bedeutung zu verhalten. Das Listing 25 zeigt die Unterstützung verschiedener H-Faktoren an einem Auszug aus der Repräsentation.

Listing 25: H-Faktoren JSON-LD

```
{
...
"@id":         "http://seminarverwaltung.de/seminare/31235",      <<LE, LO, LI, LN
"@type":       "http://seminarverwaltung.de/semantik/seminaranfrage", <<CL
"eingangsdatum":"2013-05-05",
"status":      "storniert",
"preis":       "1000",
"thema":       "Hypermedia"
...
}
```

Der Einsatz von Vorlagen ist leider nicht möglich. JSON-LD kann keine Vorlagen liefern, aus denen der Client eine URI erzeugen kann. Somit ist der Hypermedia-Faktor der Templated-Links nicht erfüllt. Die Folge davon ist, dass am Beispiel der Seminarverwaltung jede gefilterte Ansicht einen eigenen Verweis benötigt, anstatt eine für alle Ansichten gültige Vorlage auszufüllen. Ebenfalls nicht möglich ist das Nutzen von Vorlagen zum Erstellen neuer Ressourcen. Der Client benötigt spezifisches Wissen über die Art der Ressource und über die minimal benötigten Angaben um eine neue Ressource zu erzeugen. Dieses Wissen sollte in der Regel im semantischen Kontext der Applikation erfasst sein. Allerdings wird an dieser Stelle meist nur beschrieben welche Werte und Ausprägungen vorkommen können, nicht jedoch was mindestens vorhanden sein muss. JSON-LD unterstützt zwar einen Typ-Zwang, entsprechend der im Kontext festgelegten Typen, nicht jedoch eine Reglementierung der Formate, die gelesen und geschrieben werden können. Die Hypermedia-Faktoren Read- und Update-Controls sind dementsprechend nicht erfüllt. Gleichermaßen fehlt eine Unterstützung für Method-Controls. Es ist nicht möglich die auf eine Ressource zulässigen Methoden innerhalb der Nachricht festzulegen. Um eine Ressource korrekt ansprechen zu können benötigt der Client zusätzliches externes Wissen.
Analog zu den Namensräumen bei XML kann JSON-LD mit mehreren Kontext arbeiten. Wobei diese Kontext das vorhandene externe Wissen darstellen und die Grundlage für ein semantisches Verständnis bilden. Der Kontext umfasst Informationen über das verwendete Vokabular, die Bedeutung der Objekte und die möglichen Ausprägungen der Objekte. Diese Eigenschaft bildet zusammen mit der Möglichkeit komplexe Datenstrukturen darzustellen, eine gute Grundlage für den Einsatz dieses Formats in größeren, verteilten Hypermedia-Anwendungen. Ressourcen können mehrsprachig repräsentiert werden. Diese Eigenschaft ermöglicht das Erstellen einer multilingualen Anwendung. Durch die Tatsache, dass JSON-LD auch als reines JSON interpretierbar ist, eignet es sich sehr gut für den Einsatz in bestehenden JSON-Anwendungen. In den HTTP-Header muss nur ein Link-Feld eingefügt werden, sodass ein Client weiß wo sich der Kontext befindet und das er dieses Dokument als ein JSON-LD Dokument behandeln kann. Clients, die JSON-LD unterstützen, können das Format nutzen und alte Clients ohne JSON-LD Unterstützung, behandeln die Nachrichten weiterhin als reines JSON. Wie bei Collection + JSON ist JSON-LD auch ein allgemeiner Medientyp, der sich durch die Nutzung eines Kontext an eine Problemdomäne anpasst. Die Steuerung der Anwendung erfolgt hauptsächlich über die in den Nachrichten eingebetteten Hypermedia-Elemente. Tabelle 5 zeigt die Eigenschaften von JSON-LD im Überblick.

Tabelle 5: Überblick JSON-LD

H-Faktoren	**H-Level**	**Design Elemente**
LE, LO, LI, LN, CL	Level 3	Base Format = JSON, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich

3.6 JSON-Home

JSON-Home wurde entwickelt als ein Format für Web-Anwendungen, die Verweise als Grundlage der Interaktion benutzen. Design-Ziele sind unter anderem die stetige Erweiterungsmöglichkeit der Schnittstelle und eine lose Kopplung zwischen Client und Server. Diese Ziele sollen erreicht werden ohne die Notwendigkeit eine komplexe Schnittstellenbeschreibung nutzen zu müssen. Clients können die Schnittstelle durch eingebettete Verweise entdecken und benötigen dafür nur Kenntnis über die verwendeten Relationsattribute. Diese Vorgehensweise setzt voraus, dass die verwendeten Relationsattribute in umfassender und ausreichender Form dokumentiert sind. Ein JSON-Home Dokument besteht im Wesentlichen aus dem Wurzelobjekt ressources, welches Verweise auf die verfügbaren Ressourcen und die entsprechenden Relationen beinhaltet. JSON-Home unterscheidet beim Zugriff auf eine Ressource zwischen einem direkten Verweis mit dem Schlüsselwort href und einer Verweisvorlage mit dem Schlüsselwort href-template. Der direkte Verweis identifiziert hierbei immer genau eine Ressource. Die Verweisvorlage hingegen bezieht sich auf keine oder viele Ressourcen. Wenn eine Vorlage verwendet wird, muss zusätzlich eine href-vars-Eigenschaft vorhanden sein. Diese nennt einen Verweis auf eine Dokumentation, die festlegt, wie diese Vorlage genutzt werden muss und welche gültigen Werte übergeben werden können. Zusätzlich können Ressourcen mit einer hints-Eigenschaft versehen werden. Diese beinhaltet Hinweise, wie mit der Ressource interagiert werden kann. Im Kontext von JSON-Home sind diese auch nur als Hinweise zu verstehen. Sie stellen keine strikte Reglementierung dar. Die Hinweise können unter anderem folgende Ausprägungen annehmen:[33]

- allow: Beschreibt die die HTTP-Methoden, mit denen der Client auf die Ressource zugreifen kann.
- formats: Ein Hinweis auf die Formate, die von der Ressource produziert und konsumiert werden.
- accept-patch: Konsumiertes Format der Ressource wenn sie mit HTTP-PATCH angesprochen wird.
- accept-post: Konsumiertes Format der Ressource wenn sie mit HTTP-POST angesprochen wird.
- docs: Muss einen Verweis auf die Dokumentation der Ressource beinhalten.

Die Verwendung von JSON-Home im Kontext der Seminarverwaltung würde die im Listing 26 dargestellte Antwort auf eine initiale Anfrage erzeugen.

[33] Vgl. Nottingham, M. (2013), `http://tools.ietf.org/html/draft-nottingham-json-home-03`

Listing 26: Initiale Anfrage JSON-Home

```
{
"ressources": {
"http://seminarverwaltung.de/rel/seminare": {
"href":        "http://seminarverwaltung.de/seminare",
"hints":              {
"allow":              ["GET"],
"formats":            {"application/json-home":    {}}
}},

"http://seminarverwaltung.de/rel/stornierungen":  {
"href":        "http://seminarverwaltung.de/storno",
"hints":              {
"allow":              ["GET"],
"formats":            {"application/json-home":    {}}
}}
}}
```

Dem Client stehen aus seinem aktuellen Kontext heraus die zwei Ressourcen Seminare und Stornierungen zur Verfügung. Diese beiden Ressourcen sind jeweils in Objektform dargestellt und bestehen aus einer href- und hints-Eigenschaft. Href beinhaltet die URI der Ressource und hints stellt dem Client zusätzliche Informationen bezüglich der Interaktion mit der Ressource zur Verfügung. In diesem Fall liefern beide Ressourcen ein JSON-Home Dokument als Antwort auf eine Anfrage. Außerdem sollten die beiden Ressourcen nur über ein HTTP-GET angesprochen werden. Dies hat zu Folge, dass die Ressourcen nur gelesen werden können. Listing 27 zeigt eine Antwort auf die Dereferenzierung der Seminare.

Listing 27: Seminaranfragen JSON-Home

```
{
"ressources": {
"http://seminarverwaltung.de/rel/seminaranfragen": [{
"href":        "http://seminarverwaltung.de/seminare/31235",
"eingangsdatum":      "2013-05-05",
"status":             "storniert",
"preis":              "1000",
"thema":              "Hypermedia",

"hints":              {
"allow":              ["GET", "PUT"],
"formats":            {"application/json-home":    {}}
},
```

```
"http://seminarverwaltung.de/rel/seminarbeschreibung": {
"href":        "http://seminarkatalog.de/seminare/123"},
"http://seminarverwaltung.de/rel/kundenbeschreibung": {
"href":        "http://kundendb.de/kunden/567312"},
"http://seminarverwaltung.de/rel/stornierung": {
"href":        "http://seminarverwaltung.de/storno/5512"}}
],

"http://seminarverwaltung.de/rel/filter":  {
"href-template":      "http://seminarverwaltung.de/seminare/?status={status}",
"href-vars":          {"status":     " "},
"hints":              {
"allow":              ["GET"],
"formats":            {"application/json-home":    {}}
}}
}}
```

Die Repräsentation der Seminaranfragen besteht aus einem Array, das die Anfragen beinhaltet und einem Objekt, um auf die gefilterten Ansichten zugreifen zu können. Einträge des Anfragen-Arrays beinhalten neben den festgelegten Informationen wie Preis, Status und Thema eine href-Eigenschaft, die die Anfrage identifiziert. Um weitere Informationen zu einer einzelnen Anfrage zu erhalten, kann der Client diesem Verweis folgen. Die Hinweise, wie mit der Ressource interagiert werden kann, befinden sich in der hints-Eigenschaft. Die Seminaranfrage selbst konsumiert und produziert JSON-Home Dokumente und kann gelesen und beschrieben werden. Hierfür sollten entsprechend die HTTP-Methoden PUT und GET verwendet werden. Des weiteren besteht eine Seminaranfrage aus den drei eingebetteten Objekten seminarbeschreibung, kundenbeschreibung und stornierung. Die eingebetteten Objekte enthalten ausschließlich eine href-Eigenschaft. Möchte der Client mit diesen Objekten interagieren, so muss er die URI dereferenzieren und bekommt als Antwort eine Repräsentation der Ressourcen, die weitere Informationen enthalten sollte. Für eingebettete Objekte ist es nicht möglich separate Hinweise zu erstellen. Die in der Repräsentation enthaltenen Hinweise gelten nur für Ressourcen der Seminaranfrage, nicht jedoch für die eingebetteten Objekte. Neben dem Array wird ein filter-Objekt an den Client übermittelt. Das Objekt besteht seinerseits aus einem Verweis. In diesem Fall ist er kein direkter Verweis, sondern liegt in Form einer Vorlage vor, die nach Spezifikation von RFC6570 (http://tools.ietf.org/html/rfc6570) genutzt werden kann. Der Client kann einen gültigen Status an die Vorlage übergeben und dem daraus resultierenden URI per HTTP-GET folgen. Als Antwort erhält er wiederum ein JSON-Home Dokument.
JSON-Home unterstützt mit der href-Eigenschaft die Hypermedia-Faktoren Embedding- und Outbound -Links. Templated-Links werden durch die Eigenschaft href-template unterstützt. Alle so identifizierten Ressourcen können mit idempotenten und nicht-idempotenten Operationen ange-

sprochen werden. Somit sind alle fünf Link-Faktoren erfüllt. Die verwendeten Relationsbegriffe werden stets in Form einer URI angegeben und entsprechen der Unterstützung von Link-Annotaion-Controls. Read-, Update- und Method-Controls werden durch Hinweise umgesetzt, die in die Ressourcen eingebettet werden. Die Eigenschaft allow legt fest mit welcher HTTP-Methode, lesend und schreibend mit der Ressource interagiert werden kann. Listing 28 zeigt einen Ausschnitt aus einer Repräsentation mit ihren Elementen, die für eine Unterstützung der jeweiligen H-Faktoren sorgen.

Listing 28: H-Faktoren JSON-Home

```
{
...
"http://seminarverwaltung.de/rel/filter":  {              <<CL

"href-template":
"http://seminarverwaltung.de/seminare/?status={status}", <<LE, LO, LT, LI, LN

"href-vars":          {"status":     " "},
"hints":              {
"allow":              ["GET"],                            <<CM
"formats":            {"application/json-home":{}}}       <<CR, CU
...
}
```

Zudem kann mit formats festgelegt werden welcher Medientyp von der Ressource konsumiert und produziert wird. JSON-Home interpretiert die hints-Eigenschaft einer Ressource nur als Hinweis, nicht jedoch als fest definierte Regel. Das Verhalten der Applikation kann zur Laufzeit von den Hinweisen abweichen. JSON-Home eignet sich sehr gut für eine sich selbst beschreibende Schnittstelle und eine Anwendung deren Steuerung über in Nachrichten eingebettete Elemente erfolgt. Ressourcen zu identifizieren und mit Hinweisen über deren Verwendung zu versehen, ermöglicht einem Dienstkonsumenten sich selbstständig durch die Schnittstelle der Anwendung zu bewegen. Dies setzt jedoch voraus, dass die verwendeten Relationsbegriffe in einer externen Dokumentation vorhanden sind und der Client Kenntnis über deren Semantik besitzt. Dieses Vorgehen beschränkt die notwendige Dokumentation auf die verwendeten Relationen. Ein Nachteil ist das Fehlen einer Vorlage zum Erstellen von neuen Ressourcen. Diese Informationen müssen wie die genutzten Relationen in einer externen Dokumentation erfasst werden. Da als Relationswerte URIs benutzt werden, sollten sie in jedem Fall auf die dazugehörige Dokumentation zeigen. Sofern die Dienstkonsumenten nur bekannte Relationen beachten und alles andere ignorieren, kann ein JSON-Home Dokument problemlos erweitert werden. Das sorgt für eine lose Kopplung zwischen Client und Server. JSON-Home nutzt JSON als Basisformat und lässt sich durch die Verwendung von Relationsbegriffen an eine Problemdomäne anpassen. Es unterstützt die Steuerung des Appli-

kationsflusses, sowie eine Zustandsänderung der Anwendung durch eingebettete Steuerelemente. Ein wesentlicher Nachteil ist, dass eingebettete Ressourcen nicht näher beschrieben werden können. Soll eine solche Ressource verwendet werden, muss sie dereferenziert werden, um zusätzliche Informationen zu erhalten. Hinweise in einem JSON-Home Dokument liefern nur den Medientyp aber keine Aussage, welche Werte mindestens vorhanden sein müssen. Die Indikation von Fehlern ist in JSON-Home nicht möglich. Nach der Spezifikation sollen HTTP-Fehlercodes verwendet werden, anstatt dem Client ein Fehlerobjekt zu senden. Tabelle 6 zeigt die Fähigkeiten von JSON-Home im Überblick.

Tabelle 6: Überblick JSON-Home

H-Faktoren	H-Level	Design Elemente
LE, LO, LT, LI, LN, CR, CU, CL, CM	Level 3	Base Format = JSON, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich

3.7 Hypertext Application Language HAL

Die Hypertext Application Language (HAL) wurde von Mike Kelly entwickelt und ist bei der IANA offiziell in der JSON- und XML-Variante registriert. HAL wurde entwickelt um innerhalb einer Hypermedia-Anwendung, Ressourcen mit ihren Verweisen und Relationen darzustellen. HAL stellt eine Art Regelwerk zur Verfügung, sodass sich ein Entwickler nur auf die Dokumentation und Spezifikation der verwendeten Relationsbegriffe konzentrieren muss. Dabei besteht HAL im Wesentlichen aus den zwei Komponenten Ressourcen und Verweise. Ressourcen können ihren eigenen Zustand, Verweise und eingebettete Ressourcen umfassen. Auf diese Weise soll es einem Client ermöglicht werden den Verweisen entsprechend ihrer Relationen zu folgen und damit die Anwendung zu steuern. Die Wurzel eines HAL-Dokuments muss ein Ressource-Objekt sein. Das Top-Level Objekt repräsentiert immer eine Ressource. Dieses Objekt besteht aus den zwei reservierten Eigenschaften links und embedded. Weitere Eigenschaften zur Beschreibung der Ressourcen können hinzugefügt werden. Von diesen weiteren Eigenschaften wird gefordert, dass sie den aktuellen Zustand der Ressource repräsentieren und JSON konform sind. Die embedded-Eigenschaft bietet die Möglichkeit eingebettete Ressourcen darzustellen. Die Darstellung kann entweder als eine Ressource oder als ein Array von Ressourcen erfolgen. Die links-Eigenschaft beinhaltet die Relationsbegriffe mit den dazugehörigen Verweisen. Diese beziehen sich immer auf den aktuellen Kontext der Ressource. Innerhalb eines links-Objekts ist href die einzig geforderte Eigenschaft. Sie beinhaltet die URI der entsprechenden Ressource entweder als direkten Verweis oder als Vorlage. Wenn eine Vorlage verwendet wird, muss die Eigenschaft templated auf true gesetzt werden.[34]

[34] Vgl. Kelly, M. (2013), `http://tools.ietf.org/html/draft-kelly-json-hal-05`

Im Folgenden wird die Verwendung von HAL im Kontext der Seminarverwaltung gezeigt. Listing 29 zeigt die initiale Repräsentation.

Listing 29: Initiale Anfrage HAL

```
{
"_links":     {
"seminare":   {
       "href":       "http://seminarverwaltung.de/seminare",
       "type":       "application/hal+json"},

"stornierungen":     {
       "href":       "http://seminarverwaltung.de/storno",
       "type":       "application/hal+json"}}
}
```

Wie in Listing 29 dargestellt, enthält die initiale Anfrage nur ein links-Objekt. Dieses stellt dem Client die aktuell zur Verfügung stehenden Ressourcen dar. In diesem Fall sind die Relationen zusätzlich mit einer type-Eigenschaft modelliert, welche den Client über den nach der Dereferenzierung zu erwarteten Medientyp aufklärt. Dementsprechend erhält er als Antwort ein HAL-Dokument. Listing 30 zeigt die Übersicht der Seminaranfragen.

Listing 30: Seminaranfragen HAL

```
{
"seminaranfragen":   [
{
"_links":     {

"self":       {"href":      "http://seminarverwaltung.de/seminare/31235"},
},

"_embedded":  {

"seminarbeschreibung": {
"self":       {"href":      "http://seminarkatalog.de/seminare/123"},
"type":       "application/hal+json"},
"kundenbeschreibung": {
"self":       {"href":      "http://kundendb.de/kunden/567312"},
"type":       "application/hal+json"},
"stornierung":              {
"self":       {"href":      "http://seminarverwaltung.de/storno/5512"},
"type":       "application/hal+json"}
```

```
},

"eingangsdatum":      "2013-05-05",
"status":             "storniert",
"preis":              "1000",
"thema":              "Hypermedia"}],

"filter":             {
       "href":        "http://seminarverwaltung.de/seminare/?status={arg}",
       "templated":   "true"}
}
```

Die Repräsentation der Seminaranfragen besteht aus einem Array von Anfragen und einem Filter-Objekt. Das Array beinhaltet die einzelnen Seminaranfragen als Ressourcen. Eine Ressource besteht aus den JSON konformen Eigenschaften Eingangsdatum, Status, Preis und Thema, sowie den links- und embedded-Objekten. Das links-Objekt hat nur die self-Eigenschaft, welche die Ressource anhand einer URI identifiziert. So kann auf eine zusätzliche Identität in Form einer separaten Eigenschaft verzichtet werden. Würde die Liste der Seminaranfragen aus mehreren Seiten bestehen, könnte man an dieser Stelle noch weitere Verweise einfügen, die beispielsweise auf die erste und die letzte Seite der Liste zeigen. Die Anfrage selbst besitzt die eingebetteten Objekte seminarbeschreibung, kundenbeschreibung und stornierung. Die Einbettung dieser Objekte ist durch die Eigenschaft embedded kenntlich gemacht worden. In einem HAL-Dokument können diese eingebetteten Objekte mit weiteren Eigenschaften genauer beschrieben werden. Die type-Eigenschaft gibt Auskunft darüber, welchen Medientyp der Client nach einer Dereferenzierung zu erwarten hat. Eine Identifizierung des Objektes selbst ist durch die self-Eigenschaft umgesetzt. Ob ein eingebettetes Objekt vollständig, partiell oder erst auf Anfrage dereferenziert wird, liegt in der Verantwortung des Dienstkonsumenten. Unabhängig von diesem Beispiel ist es in einem HAL-Dokument möglich, die eingebetteten Objekte mit allen für eine Ressource gültigen Eigenschaften zu versehen. Auch können diese Objekte weitere eingebettete Objekte beinhalten.
HAL stellt einen sehr generischen Rahmen für einen Medientyp zur Verfügung. Bei der Entwicklung einer Hypermedia-Anwendung, die HAL als Medientyp einsetzt, besteht die wichtigste Aufgabe darin, alle verwendeten Relationsbegriffe umfassend zu dokumentieren. Die Identifikation einer Ressource erfolgt über ihre self-Eigenschaft und der dazugehörigen Pfilchtangabe einer URI. HAL unterstützt alle Link-Faktoren. Embedding- und Outbound-Links sind jeweils mit einer href-Eigenschaft versehen. Diese Verweise können mit idempotenten und nicht idempotenten Methoden angesprochen werden. HAL erfüllt damit auch die Link-Faktoren Idempotent- und Non-Idempotent-Links. Vorlagen werden mit der templated-Eigenschaft kenntlich gemacht und realisieren den Hypermedia-Faktor Templated-Links. Der Hypermedia-Faktor der Link-Annotation-Controls wird durch die Verwendung von Relationsbegriffen ermöglicht. Die Relationsbegriffe müssen in einer externen Dokumentation festgehalten werden oder es werden offiziell definierte Re-

lationen verwendet. Eine kleine Sammlung offiziell definierter Relationen hat die IETF unter dem RFC5988 veröffentlicht. Listing 31 zeigt die unterstützten H-Faktoren an einem Ausschnitt einer Repräsentation.

Listing 31: H-Faktoren HAL

```
"seminarbeschreibung": {                                          <<CL
"self":
{"href":       "http://seminarkatalog.de/seminare/123"},          <<LE, LO, LI, LN
"type":        "application/hal+json"}
...
"filter":             {                                           <<CL
"href": "http://seminarverwaltung.de/seminare/?status={arg}",     <<LT
"templated":   "true"}
```

Read-, Update- und Method-Controls werden nicht unterstützt. Eine Vorlage zum Anlegen neuer Ressourcen kann HAL nicht zur Verfügung stellen. Der produzierte und konsumierte Medientyp kann zwar über die type-Eigenschaft erfasst werden, es fehlen jedoch weitere Informationen über die geforderten Inhalte. Diese müssen somit Bestandteil einer externen Dokumentation sein. Eine Besonderheit von HAL ist, dass eingebettete Ressourcen partiell dargestellt werden können oder erst auf Anfrage dereferenziert werden. Das embedded-Objekt ermöglicht zudem eine beliebig komplexe Verschachtelung der beinhalteten Ressourcen. Eingebettete und normale Ressourcen werden gleichwertig behandelt und können dementsprechend mit allen verfügbaren Eigenschaften versehen werden. Dem Client kann es so ermöglicht werden, den Medientyp einer eingebetteten Ressource bereits vor dem Auflösen des Verweises zu kennen. HAL bietet keine Möglichkeit das konsumierte oder produzierte Format exakt zu spezifizieren. Auch fehlt die Möglichkeit, eine zulässige Methode zum Interagieren mit einer Ressource festzulegen. Tabelle 7 zeigt die Möglichkeiten von HAL im Überblick.

Tabelle 7: Überblick HAL

H-Faktoren	H-Level	Design Elemente
LE, LO, LT, LI, LN, CL	Level 3	Base Format = JSON, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich

3.8 Siren

Siren wurde von Kevin Swiber entwickelt und ist bei der IANA als Medientyp registriert. Siren versteht sich als eine Art generische Spezifikation. Die Umsetzung von Siren kann sowohl in JSON als auch in XML erfolgen. Im Kontext von Siren spricht man von Klassen und Entitäten, wobei die Entitäten analog zu dem Begriff der Ressourcen verwendet werden. Das Ziel von Siren ist es, Informationen über den aktuellen Zustand einer Ressource, Operationen auf eine Ressource und Verweise zu Navigationszwecken für einen Dienstkonsumenten verfügbar zu machen. Eine Entität definiert Siren als eine adressierbare Ressource mit Eigenschaften und Operationen, sowie Sub-Entitäten und Verweisen zur Navigation. Entitäten und Sub-Entitäten können in einem Siren Dokument gleichwertig behandelt und im gleichen Umfang mit Eigenschaften versehen werden. Siren unterscheidet zwei Arten von Sub-Entitäten, abhängig davon ob ein href-Wert vorhanden ist oder nicht. Ist ein href-Wert vorhanden, wird die Entität als eingebetteter Verweis behandelt. Andernfalls als eine eingebettete Repräsentation einer Entität. Sub-Entitäten müssen im Gegensatz zu einer normalen Entität ein Relationsbegriff beinhalten, der die Beziehung zu der Eltern-Entität beschreibt. Die class-Eigenschaft ist für Entitäten und Sub-Entitäten definiert als ein String, der die Art und Natur einer Entität beschreibt. Grundsätzlich ist die class-Eigenschaft abhängig von der Implementierung und muss in einer Dokumentation beschrieben werden. Die properties-Eigenschaft ist definiert als eine Sammlung von Schlüssel-Wert Paaren, die etwas über den aktuellen Zustand einer Entität aussagen. Verweise dienen in Siren hauptsächlich der Navigation. Entitäten sollten zur Identifikation einen Verweis auf sich selbst beinhalten. Dieser Verweis wird mit dem Relationsbegriff self kenntlich gemacht. Die actions-Eigenschaft stellt ein Array von möglichen Operationen der Entität zur Verfügung. Die minimale Definition einer Operation muss eine name-Eigenschaft zur Identifizierung und eine href-Eigenschaft mit der angesprochenen URI beinhalten. Die class-Eigenschaft einer Operation kann dazu genutzt werden, um semantische Informationen hinzuzufügen. Zusätzlich kann eine Aktion mit einer method-Eigenschaft versehen werden, die eine entsprechende Protokoll-Methode deklariert und einer type-Eigenschaft, die den zu erwarteten Medientyp festlegt. Die fields-Eigenschaft kann innerhalb einer Aktion als Vorlage interpretiert werden. Sie gibt Auskunft darüber, welche Werte von der Operation gefordert werden, um erfolgreich ausgeführt zu werden.[35]
Angewendet auf das Fallbeispiel der Seminarverwaltung ergibt sich die in Listing 32 dargestellte Antwort auf eine initiale Anfrage.

Listing 32: Initiale Anfrage Siren+JSON

```
{
"class":       ["seminarverwaltung"],
"links":       [
{"rel":        ["seminare"],
"href":        "http://seminarverwaltung.de/seminare"},
```

[35] Vgl. Swiber, K. (2013), https://github.com/kevinswiber/siren

```
{"rel":        ["stornierungen"],
"href":        "http://seminarverwaltung.de/storno"}
]}
```

Die class-Eigenschaft gibt in diesem Fall Auskunft darüber, an welcher Stelle sich der Client befindet. Angenommen der Begriff Seminarverwaltung ist in einer Dokumentation als die Wurzel der Schnittstelle definiert, kann der Client diese Repräsentation als einen Einstiegspunkt in die Anwendung verwenden. Die verwendeten Relationsbegriffe Seminare und Stornierungen stellen die Beziehung zwischen der aktuellen Position des Clients und den zur Verfügung gestellten Verweisen dar. In einem Siren Dokument werden die class- und rel-Eigenschaft stets in Form eines Arrays angegeben. Dies ermöglicht die Verwendung mehrerer Relationsbegriffe zur semantischen Beschreibung. Ein Verweis kann in einem Siren Dokument nicht näher beschrieben werden, da die einzig gültigen Eigenschaften rel und href sind. Wenn der Client den Verweis auf die Seminare dereferenziert, bekommt er die in Listing 33 dargestellte Antwort.

Listing 33: Seminaranfragen Siren+JSON

```
{
"class":       ["anfragen"],

"entities":    [
{
"class":       ["seminaranfrage"],
"rel":         ["anfrage"],

"properties":  {
"eingangsdatum":      "2013-05-05",
"status":              "storniert",
"preis":               "1000",
"thema":               "Hypermedia"
},

"entities" [
{"rel":        "seminar",
"href":        "http://seminarkatalog.de/seminare/123"},

{"rel":        "kunde",
"href":        "http://kundendb.de/kunden/567312"},

{"rel":        "stornierung",
"properties": {
"storno-nr":   "5512",
```

```
"stornierungsdatum": "2013-06-01"},
"links":       [{
"rel":         "self",
"href":        "http://seminarverwaltung.de/storno/5512"}]}
],

"links":       [{
"rel":         "self",
"href":        "http://seminarverwaltung.de/seminare/31235"}]
}
],

"actions":     [
{
"name":        "add-seminar",
"method":      "PUT",
"href":        "http://seminarverwaltung.de/seminare",
"type":        "application/vnd.siren+json",
"fields":      [
{"name":       "preis",        "type":       "number"},
{"name":       "thema",        "type":       "text}]
}],

"links":       [
{"rel": ["filter-nd"],
"href": "http://seminarverwaltung.de/seminare/?status=nd"}
]}
```

Die in Listing 33 dargestellte Repräsentation enthält ein Array der aktuellen Seminaranfragen, eine actions-Eigenschaft und eine links-Eigenschaft. Die class-Eigenschaft am Anfang des Dokuments beschreibt semantisch den Inhalt der Repräsentation. Die repräsentierten Objekte befinden sich innerhalb der entities-Eigenschaft. Die optionale Angabe der class-Eigenschaft definiert einen Eintrag als Seminaranfrage und die rel-Eigenschaft beschreibt die Beziehung zur eigentlichen Repräsentation der Seminaranfragen. Eine properties-Eigenschaft ist ebenfalls eine optionale Angabe und definiert den aktuellen Zustand der Entität. Ihr ist unter anderem zu entnehmen, dass die Anfrage storniert wurde und Preis 1000 Geldeinheiten betragen sollte. Um diese Angabe auswerten zu können, benötigt der Client Wissen über die verwendeten Relationsbegriffe und deren Bedeutung. Neben der Beschreibung des aktuellen Zustandes, enthält die Entität noch die drei Sub-Entitäten Seminar, Kunde und Stornierung. Seminar und Kunde sind in diesem Beispiel modelliert als eingebettete Verweise. Der Client entscheidet darüber, zu welchem Zeitpunkt sie dere-

ferenziert werden. Anders verhält es sich bei der Sub-Entität Stornierung. Sie ist als eingebettete Entität dargestellt. Eingebettete Entitäten sind wie vollwertige Entitäten zu behandeln und können ebenfalls mit allen für Entitäten gültigen Eigenschaften versehen werden. Wie die Eltern-Entität verfügt diese über ein links-Array, welches den URI auf das Objekt selbst enthält. Wenn diese Eigenschaft von allen Entitäten gefordert wird, kann die self-Eigenschaft zur Identifikation der Entität benutzt werden. Neben dem Array von Seminaranfragen verfügt die Repräsentation über eine actions-Eigenschaft, die beschreibt welche Operationen aus dem aktuellen Kontext heraus angeboten werden. Entsprechend der Nachricht, besteht die Möglichkeit eine weitere Seminaranfrage hinzuzufügen. Eine Operation muss nach der Spezifikation mindestens eine name-Eigenschaft mit der semantischen Bedeutung und eine href-Eigenschaft mit der angesprochenen URI enthalten. Die anderen Angaben sind optional aber sehr wertvoll, wenn es darum geht die Interaktion genauer zu beschreiben. Die method-Eigenschaft nennt die gültige Methode, über die mit der Ressource interagiert werden kann. Der konsumierte Medientyp kann mittels der type-Eigenschaft festgelegt werden. Zusätzlich kann eine Aktion mit einer Art Vorlage, dem fields-Array, versehen werden. Dieses Array beschreibt, welche Werte mindestens vorhanden sein müssen um die Aktion erfolgreich durchzuführen. Zuletzt verfügt die Repräsentation noch über ein weiteres links-Array, das der Client zur Navigation verwenden kann. Beispielhaft enthält es hier einen Verweis auf eine der gefilterten Ansichten.
Embedded- und Outbound-Links werden an verschiedenen Stellen unterstützt und können zur Identifikation einer Entität verwendet werden. Die so identifizierten Entitäten können mit idempotenten und nich-idempotenten Operationen angesprochen werden. Siren ermöglicht es, Verweise mit Relationsbegriffen semantisch anzureichern. Link-Annotation-Controls werden auf diese Weise umgesetzt. Die semantische Bedeutung der verwendeten Relationsbegriffe muss in einer externen Dokumentation erfasst werden. Eine positive Eigenschaft ist, dass eine Ressource mit dem actions-Array, Operationen genau spezifizieren und dem Client zur Verfügung stellen kann. Die Spezifikation erfolgt innerhalb des actions-Array durch die Identifikation der Operation über einen Namen, die gültigen Methoden, den angesprochenen URI und den konsumierten Medientyp. Zusätzlich kann eine Aktion mit einem fields-Array ausgestattet werden. Dieses informiert den Client über die mindestens benötigten Werte der Aktion. Das Zusammenspiel aus actions-Array und dem beinhalteten fields-Array, stellt dem Client alle Informationen zur Verfügung, die er benötigt um mit der Ressource zu interagieren und erfüllt den Hypermedia-Faktor Update-Controls und Method-Controls. Listing 34 zeigt die Umsetzung der H-Faktoren beispielhaft an einem Auszug aus einer Repräsentation.

Listing 34: H-Faktoren Siren+JSON

```
{
...
"actions":     [
{
"name":        "add-seminar",                    <<CL
"method":      "PUT",                            <<CM
```

```
"href": "http://seminarverwaltung.de/seminare",    <<LE, LO, LI, LN
"type": "application/vnd.siren+json",              <<CU
"fields":      [
{"name":       "preis",        "type":       "number"},
{"name":       "thema",        "type":       "text}]
}]
...
}
```

Die Möglichkeit Entitäten mit gleichwertigen Sub-Entitäten und Informationen zum aktuellen Zustand auszustatten, bietet einen umfassenden semantischen Kontext auch für komplexere Datenstrukturen. Da der Client entscheidet wann ein eingebetteter Verweis dereferenziert wird, können durchaus auch partielle Anfragen gestellt werden. Leider kann Siren keine Templated-Links darstellen und im Beispiel der Seminarverwaltung müssen die gefilterten Ansichten als absolute Verweise an den Client übergeben werden. Eine Umsetzung der Read-Controls ist ebenfalls nicht vorhanden. Der Client wird nur informiert, welches Format er zu erwarten hat, er kann aber kein bestimmtes Format anfordern. Tabelle 8 zeigt die Eigenschaften von Siren im Überblick.

Tabelle 8: Überblick Siren

H-Faktoren	H-Level	Design Elemente
LE, LO, LI, LN, CU, CM, CL	Level 3	Base Format = JSON, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich

3.9 Atom

Atom ist ein offizieller IETF-Standard aus dem Jahr 2005. Er ist bei der IANA registriert als application/atom+xml. Als Basisformat kommt XML zum Einsatz und ein Atom Dokument muss stets einem wohlgeformten XML Dokument entsprechen. Der ursprüngliche Nutzen von Atom liegt in der Verbreitung von Web-Ressourcen, wie Blogs oder Schlagzeilen der Nachrichten. Ein Atom Dokument besteht im wesentlichen aus feed-Elementen und entry-Elementen. Alle in Atom verwendeten Elemente stammen aus dem Namensraum `http://www.w3.org/2005/atom`. Zusätzlich kann ein Atom Dokument weitere, selbst entworfene Namensräume enthalten. Ein feed-Element besteht aus Metadaten, die den Feed beschreiben und einer beliebigen Anzahl von entry-Elementen. Die Spezifikation sieht die drei Pflichtelemente id, title und updated für ein feed-Element vor. Id sollte eine URI zur Identifikation beinhalten, title den Titel des Feeds und updated das Datum der letzten Änderung. Für ein entry-Element werden die gleichen drei Pflichtangaben gefordert. Atom verfügt unter anderem über ein Konstrukt für Verweise, dem link-Element. Das link-Element kann

einem feed- und entry-Element hinzugefügt werden. Das link-Element verfügt neben einem href-Attribut, welches eine URI auf eine Ressource beinhaltet, über weitere Attribute zur Beschreibung des Verweises.[36]
Im Jahr 2007 definierte die IETF das Atom Publishing Protocol (AtomPub). Dieses Protokoll soll auf Basis von HTTP, Atom Dokumente übertragen und Möglichkeiten zur Erstellung und Bearbeitung von Ressourcen zur Verfügung stellen. Das Atom Publishing Protocol definiert hierfür die drei Einrichtungen Collections, Services und Editing. Eine Collection ist definiert als eine Sammlung von Ressourcen, wobei eine Ressource entweder ein Datenobjekt oder ein Dienst sein kann. AtomPub unterscheidet 2 Arten von Ressourcen. Eine entry-Ressource verfügt über eine Repräsentation im Atom-Format, wohingegen eine Media-Ressource in einem beliebigen anderem Format vorliegen kann. Services dienen dem Auffinden und Beschreiben von Collections. Um Statusinformationen zu übertragen, verwendet AtomPub die Response-Codes von HTTP. Um das AtomPub nutzen zu können, muss der Namensraum `http://www.w3.org/2007/app` in die Instanz eingebunden werden. Dieser Namensraum definiert unter anderem das Service-Dokument. Ein solches Dokument kann verwendet werden um einen Client über die Möglichkeiten und verfügbaren Ressourcen zu informieren. Das Service-Dokument ist als application/atomsvc+xml bei der IANA registriert. Dieses Dokument besteht aus einem Workspace-Element, welches mehrere Collections gruppieren kann. Alle Dokumente dieses Protokolls müssen als wohlgeformtes XML erstellt werden.[37]
Auf eine detaillierte Beschreibung weiterer von Atom und AtomPub spezifizierten Elemente, Konstrukte und Attribute wird an dieser Stelle bewusst verzichtet. Auf den ersten Blick scheint Atom nicht unbedingt geeignet für den Einsatz in einer verteilten Hypermedia-Umgebung. Wenn die Spezifikation jedoch genauer betrachtet wird, fällt auf, dass sie sehr generisch ist. Atom kann mehrere Einträge abbilden, diese können Verweise enthalten und die Verweise können mit weiteren semantischen Informationen versehen werden. Neben den bereits definierten Begriffen kann Atom durch den Einsatz von zusätzlichen Namensräumen erweitert werden. Über Service-Dokumente kann ein Client durch die Anwendung geleitet werden und es stehen Möglichkeiten zur Verfügung, um die Interaktion mit einer Ressource zu beschreiben. Diese Fakten entsprechen den Anforderungen an einen Medientyp, der in einer Hypermedia-Umgebung eingesetzt werden soll. Angenommen Atom kommt als Medientyp in Verbindung mit HTTP für das Fallbeispiel Seminarverwaltung zum Einsatz. Listing 35 zeigt die Antwort auf eine initiale Anfrage in Form eines Service-Dokuments.

Listing 35: Initiale Anfrage Atom Service

```
<?xml version="1.0" encoding='utf-8'?>
       <service xmlns="http://www.w3.org/2007/app"
                      xmlns:atom="http://www.w3.org/2005/Atom">
<workspace>
      <atom:title>Seminarverwaltung API-Root</atom:title>
```

[36] Vgl. Nottingham, M. (2005), `http://tools.ietf.org/html/rfc4287`
[37] Vgl. Gregorio, J. (2007), `http://tools.ietf.org/html/rfc5023`

```
      <collection
          href="http://seminarverwaltung.de/seminare">
        <atom:title>Seminaranfragen</atom:title>
        <accept>application/atom+xml</accept>
      </collection>
      <collection
          href="http://seminarverwaltung.de/storno" >
        <atom:title>Stornierungen</atom:title>
        <accept>application/atom+xml</accept>
      </collection>
    </workspace>
     </service>
```

Listing 36 zeigt eine alternative Darstellung, bei der die Collections als Einträge eines Feeds umgesetzt sind.

Listing 36: Initiale Anfrage Atom-Feed

```
<?xml version="1.0" encoding='utf-8'?>
       <feed xmlns:app="http://www.w3.org/2007/app"
                     xmlns="http://www.w3.org/2005/Atom">
   <title>Seminarverwaltung API-Root</title>
      <app:collection
          href="http://seminarverwaltung.de/seminare">
        <title>Seminaranfragen</title>
        <app:accept>application/atom+xml</app:accept>
      </app:collection>
      <app:collection
          href="http://seminarverwaltung.de/storno" >
        <title>Stornierungen</title>
        <app:accept>application/atom+xml</app:accept>
      </app:collection>
       </feed>
```

Beide Dokumente enthalten aus der Sicht eines Clients äquivalente Informationen. Sie unterscheiden sich nur in dem verwendeten Default-Namensraum und darin, ob eine Collection in einem Feed oder einem Workspace eingebettet wird. Abstrakt gesehen wird von beiden Dokumenten die gleiche Information über einen Dienst zur Verfügung gestellt. Das Service-Dokument beinhaltet einen Workspace und zwei Collections, die auf verfügbare Ressourcen verweisen. Außerdem beschreibt das accept-Element den von der Ressource konsumierten Medientyp. Möchte der Client eine neue Ressource erstellen, muss ein POST mit dem akzeptierten Format an den URI gesendet werden. Als Antwort auf eine Dereferenzierung der ersten Collection, erhält der Client

die in Listing 37 dargestellte Repräsentation der Seminaranfragen.

Listing 37: Seminaranfragen Atom-Feed

```
<?xml version="1.0" encoding='utf-8'?>
        <feed xmlns:app="http://www.w3.org/2007/app"
                 xmlns="http://www.w3.org/2005/Atom"
                 xmlns:sem="http://seminarverwaltung.de/ns">
    <title>Seminaranfragen</title>
      <id>http://seminarverwaltung.de/seminare</id>

          <entry>
          <title>Seminaranfrage</title>
               <id>http://seminarverwaltung.de/seminare/31235</id>

               <sem:eingang>2013-05-05</sem:eingang>
               <sem:status>storniert</sem:status>
               <sem:preis>1000</sem:preis>
               <sem:thema>Hypermedia</sem:thema>

          <link href="http://seminarkatalog.de/seminare/123"
          rel="seminar" type="application/atom+xml"/>

          <link href="http://kundendb.de/kunden/567312"
          rel="kunde" type="application/atom+xml"/>

          <link href="http://seminarverwaltung.de/storno/5512"
           rel="stornierung" type="application/atom+xml"/>
          </entry>

          <app:collection
          href="http://seminarverwaltung.de/storno" >
        <title>Stornierungen</title>
        <app:accept>application/atom+xml</app:accept>
      </app:collection>

      <link href="http://seminarverwaltung.de/seminare/?status=nd"
           rel="status-nd" type="application/atom+xml"/>

       </feed>
```

Als Antwort bekommt der Client ein Atom Dokument, dass um einen neuen Namensraum mit dem Präfix sem erweitert wurde. Dieser neue Namensraum enthält die Definitionen der verwendeten Relationsbegriffe, die nicht aus der Dokumentation von Atom stammen. Die Ressource Seminaranfragen wird über ein id-Element identifiziert und besteht aus einem Feed, der die einzelnen Anfragen enthält. Die Identifikation eines Eintrags erfolgt über das id-Element. Dieses beinhaltet den URI des Eintrags selbst. Informationen bezüglich des aktuellen Zustands werden durch Elemente aus dem problembezogenen Namensraum `http://seminarverwaltung.de/ns` abgebildet. Das Hinzufügen von Namensräumen ermöglicht es, das allgemeine Format Atom an eine sehr spezifische Problemdomäne anzupassen. In dieser Nachricht stammen die Elemente eingang, status, preis und thema aus dem selbst definierten Namensraum.
Die mit der Anfrage verknüpften Objekte sind über link-Elemente eingebunden. Jedes link-Element beinhaltet einen Relationsbegriff und ein type-Attribut, dass Auskunft darüber gibt, was für ein Medientyp bei einer Dereferenzierung zurück geliefert wird. Neben den Einträgen existiert noch eine Collection, die den Client darüber informiert, wie er auf die Stornierungen zugreifen kann und an welcher Stelle sie gefunden werden können. Das accept-Element besagt, dass der Client zum Anlegen einer neuen Stornierung, ein Atom Dokument an den angegebenen URI mittels eines HTTP-POST übertragen muss. Die Methode HTTP-POST ist für das Anlegen und Aktualisieren einer Ressource festgeschrieben und muss entsprechend der Spezifikation verwendet werden. Das Benutzen einer alternativen Methode wie HTTP-PUT ist nicht möglich. Der Feed selbst beinhaltet noch ein link-Element, mit einem Verweis auf eine gefilterte Ansicht der Seminarverwaltung. Auch an dieser Stelle ist wieder ein Relationsbegriff verwendet, um den Verweis mit semantischen Informationen anzureichern.
Outbound-Links können in Atom Dokumenten entweder durch ein link- oder ein content-Element umgesetzt werden. Wenn ein Verweis eingebettet werden soll, stellt Atom beispielsweise das content-Element zur Verfügung. Das content-Element unterscheidet einen eingebetteten und einen externen Verweis durch das verwendete Attribut. Hierbei realisiert href einen externen und src einen eingebetteten Verweis. Das content-Element wir an dieser Stelle verwendet, weil ein das link-Element keine Unterstützung für das src-Attribut bietet. Innerhalb eines Atom Dokuments werden eingebettete Verweise sofort dereferenziert und die Informationen werden in das Dokument eingefügt. Die enthaltenen Verweise können mit idempotenten und nicht-idempotenten Operationen angesprochen werden, was die Hypermedia-Faktoren LN und LI erfüllt. Atom ermöglicht das Anreichern von Verweisen mit zusätzlichen Informationen. Diese können entweder aus den bereits in der Spezifikation definierten Relationsbegriffen stammen oder selbst festgelegt werden. Dadurch erfüllt Atom den Hypermedia-Faktor Link-Annotation-Controls. Das accept-Element legt innerhalb einer Collection fest, welcher Medientyp von der angesprochenen Ressource konsumiert wird. Der Client hat somit Kenntnis darüber, mit welchem Medientyp er eine neue Ressource anlegen und eine bestehende aktualisieren kann. Der Control-Update Hypermedia-Faktor wird dementsprechend erfüllt. Listing 38 zeigt die die umgesetzten H-Faktoren an einem Beispiel.

Listing 38: H-Faktoren Atom

```
...
<app:collection
    href="http://seminarverwaltung.de/storno" >                    <<LO, LI, LN
  <title>Stornierungen</title>
  <app:accept>application/atom+xml</app:accept>                    <<CU
</app:collection>

  <link href="http://seminarkatalog.de/seminare/123"
      rel="seminar"                                                <<CL
      type="application/atom+xml"/>

<content src="http://seminarverwaltung.de/storno/5512"/>           <<LE, LI, LN
...
```

Das Hinzufügen von Namensräumen bietet die Möglichkeit, neue Elemente zu erstellen und Atom an eine Problemdomäne anzupassen. Leider ist Atom an dieser Stelle auf HTTP-POST beschränkt und kann keine idempotenten Operationen zum Schreiben auf Ressourcen anbieten. Control-Read wird von Atom nicht umgesetzt. Der Client hat nicht die Möglichkeit aus einem Atom-Dokument heraus, einen von ihm bevorzugten Medientyp als Repräsentation anzufordern. Ein Nachteil von Atom ist die fehlende Unterstützung von Vorlagen. Nach Spezifikation ist es nicht möglich, ein URI-Template in ein Atom Dokument einzufügen. Der Hypermedia-Faktor LT wird nicht unterstützt. Das hat für den Einsatz innerhalb der Seminarverwaltung zur Folge, dass alle gefilterten Ansichten als separate Verweise eingefügt werden müssen. Das Anlegen von neuen Ressourcen kann Atom mit der verwendeten HTTP-Methode und dem konsumierten Medientyp zwar grob beschreiben, aber nicht exakt festlegen. Beispielsweise fehlt eine Vorlage, die definiert, welche Werte in einer gültigen Repräsentation mindestens vorhanden sein müssen. Atom stellt ein sehr generisches Format mit vielen Anpassungsmöglichkeiten zur Verfügung. Das Verwenden von Relationsbegriffen und die Möglichkeit verschiedene Namensräume einzubinden, lassen einen hohen Grad semantischer Anreicherung zu. Tabelle 9 zeigt die Eigenschaften von Atom im Überblick.

Tabelle 9: Überblick Atom

H-Faktoren	H-Level	Design Elemente
LE, LO, LI, LN, CU, CL	Level 3	Base Format = XML, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich

3.10 XHTML

Dieses Unterkapitel beschreibt die Verwendung von XHTML in einer verteilten Hypermedia- Anwendung. Da XTHML in dem Fallbeispiel der Seminarverwaltung zur Kommunikation von Maschinen untereinander eingesetzt werden soll, erhält es den Vorzug gegenüber HTML. XHTML kann auch als ein reines, wohlgeformtes XML Dokument betrachtet und verarbeitet werden. Dementsprechend ist es für Maschinen leicht lesbar und es existieren bereits viele Parser für die verschiedensten Sprachen. Andere bereits vorgestellte Medientypen verwenden in der Regel ein Basisformat ohne Hypermedia-Eigenschaften. Reines JSON oder XML sieht keine Verwendung von Verweisen, Vorlagen oder semantischen Informationen vor. Allgemein betrachtet sind diese beiden Basisformate also eine Sammlung von Schlüssel-Wert-Paaren ohne weitere Bedeutung. Um sie trotzdem in einer Hypermedia-Anwendung einsetzen zu können, werden sie mit zusätzlichen Fähigkeiten und Eigenschaften erweitert. Diese Erweiterung erfolgt in der Regel mittels einer Spezifikation, die das Basisformat um definierte Schlüsselbegriffe erweitert. Als Alternative kann ein Medientyp wie XHTML verwendet werden, der bereits viele der geforderten Hypermedia-Eigenschaften umfasst. Die wohl gebräuchlichste Form von XHTML, ist die Darstellung von Dokumenten innerhalb eines Browsers.
XHTML wurde ursprünglich entwickelt um verschiedene Arten von Dokumenten im Internet darzustellen und besteht aus vielen Elementen, die der Strukturierung von Text dienen. Jedes XHTML Dokument muss auch als wohlgeformtes XML Dokument interpretiert werden können. Die Spezifikation von XHTML definiert eine Reihe von Modulen, die innerhalb eines Dokuments verwendet werden können. Diese Module stellen eine Definition der benutzbaren Elemente und deren Attribute dar. So beschreibt die Spezifikation von XHTML unter anderem die Verwendung von Verweisen, Ressourcen, semantischen Informationen, Strukturelementen und vieles mehr.[38]
Angewendet auf das Fallbeispiel Seminarverwaltung, könnte der Client die in Listing 39 dargestellte Antwort auf eine initiale Anfrage erhalten. Repräsentiert wird der Einstiegspunkt der Anwendung.

Listing 39: Initiale Anfrage XHTML

```
<?xml version="1.0" encoding="utf-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Seminarverwaltung</title>
  </head>
  <body>
    <link rel="Seminaranfragen"
        href="http://seminaranfragen.de/seminare"
        hreftype="application/xhtml+xml"
        hreflang="de"></link>
```

[38] Vgl. Birbeck, M. u.a. (2010), http://www.w3.org/TR/xhtml2/xhtml2.html

```
    <link rel="Stornierungen"
        href="http://seminaranfragen.de/storno"
        hreftype="application/xhtml+xml"
        hreflang="de"></link>
   </body>
```

Bereits die Repräsentation des Einstiegspunktes, kann mit vergleichsweise vielen semantischen Informationen angereichert werden. Neben dem üblichen, HTML-konformen Rahmen enthält das Dokument zwei link-Elemente. Das rel-Attribut beschreibt die Semantik des Verweises und muss in einer Dokumentation anwendungsspezifisch beschrieben werden. Die drei weiteren Attribute sind bereits durch den Medientyp application/xhtml+xml spezifiziert. Ein URI der Ressource liegt innerhalb des href-Attributes. Hreftype enthält Informationen über den zu erwarteten Medientyp nach der Dereferenzierung des Verweises. Die verwendete, natürliche Sprache wird mittels des hreflang-Attributes beschrieben. Listing 40 zeigt die Repräsentation der Seminaranfragen.

Listing 40: Seminaranfragen XHTML

```
<?xml version="1.0" encoding="utf-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Seminaranfragen</title>
  </head>
  <body>
  <ul class="Liste Anfragen">
  <li class="Anfrage">

  <link rel="self" href="http://seminaranfragen.de/seminare">
  </link>

        <span class="eingangsdatum">2013-05-05</span>
         <span class="status">storniert</span>
        <span class="preis">1000</span>
        <span class="thema">Hypermedia</span>
        <span class="stornierung" src="http://seminarverwaltung.de/storno/5512"
        srctype="application/xhtml+xml"></span>

  <div class="seminar">
  <a href="http://seminarkatalog.de/semianre/123"></a>
  </div>
  <div class="kunde">
  <a href="http://kundendb.de/kunden/567312"></a>
```

```
  </div>
  </li>
  </ul>
  <form name="neue-anfrage" method="post"
        action="http://seminaranfragen.de/seminare">
  <input type="text" name="preis" />
  <input type="text" name="thema" />
  <input type="submit" value="Submit"/>
  </form>
  <form name="filter" method="get" action="http://seminarverwaltung.de/seminare/" >
    <select name="status">
    <option>eingang</option>
    <option>bearbeitung</option>
    <option>bestaetigt</option>
    <option>erledigt</option>
    <option>nd</option>
    </select>
</form>
   </body>
```

Die Repräsentation der Seminaranfragen besteht im Wesentlichen aus zwei Formularen und einer ungeordneten Liste, die Seminaranfragen beinhaltet. Das ul-Element kann mit einem class-Attribut versehen werden. Das class-Attribut beschreibt nach der Spezifikation stets einen semantischen Kontext und kann für verschiedene Elemente genutzt werden. In diesem Fall ist die Liste als eine Liste der Anfragen und ein Eintrag als eine Anfrage beschrieben. Der Eintrag wird durch das class-Attribut Anfrage gekennzeichnet und besteht aus einem Verweis zur Identifikation sowie span- und div-Elementen. Diese sind in der XHTML-Spezifikation definiert als generische Mechanismen, um ein Dokument zu strukturieren. Die allgemeine Spezifikation ermöglicht es, die div- und span-Elemente mit dem class-Attribut an eine Problemdomäne anzupassen. So ist in diesem Fall ein Eintrag als Anfrage definiert. Die Semantik der span-Elemente mit den Klassen eingangsdatum, status, preis und thema stammen aus der anwendungsspezifischen Problemdomäne und stellen den aktuellen Zustand einer Anfrage dar. Das span-Element stornierung stellt die zur Anfrage gehörende Stornierung, als eingebetteten Verweis zur Verfügung. Die Einbettung erfolgt hierbei durch das Verwenden eines src-Attributes. Das src-Attribut verweist auf den URI der Ressource und das srctype-Attribut beinhaltet den Medientyp der Repräsentation.
Die beiden anderen, mit dem Eintrag verknüpften Ressourcen Seminar und Kunde, sind mittels div-Elementen und einer semantischen Beschreibung durch das class-Attribut in die Struktur des Dokumentes eingebunden. Sie beinhalten jeweils ein a-Element mit dem Verweis auf die Ressource. A-Elemente stellen in Kombination mit dem href-Attribut einen externen Verweis dar. Zusätzlich beinhaltet diese Repräsentation zwei Formulare, die dem Anlegen einer neuen Ressource

und dem Zugriff auf die gefilterten Ansichten der Semiarverwaltung dienen. Das erste Formularelement besteht aus den Attributen name, method und action. Diese informieren den Client über den Namen des Formulars, die verwendete Methode und den URI, der zum Anlegen einer neuen Ressource angesprochen werden muss. Die input-Elemente legen fest, welche Werte der Client übergeben muss und von welchem Typ diese Werte sein müssen. Das Formular stellt somit eine Art Vorlage dar, woraus der Client ablesen kann, wie eine neue Anfrage angelegt wird. Im Kontext der Formulare sollte das name-Attribut Auskunft über die Bedeutung des Formulares geben. Wie alle anderen semantischen Informationen, muss diese Bedeutung in einer Dokumentation erfasst werden. Das zweite Formular kann von einem Dienstkonsumenten genutzt werden, um auf die gefilterten Ansichten der Seminarverwaltung zuzugreifen. Die Attribute verhalten sich analog zum ersten Formular, mit dem Unterschied, dass an dieser Stelle die Methode GET zum Einsatz kommt. Ein Client kann einen in der Dokumentation definierten Status an das Formular übergeben und kann einen entsprechenden URI erzeugen. Wird der Status nd übergeben, resultiert aus dem Formular der URI `http://seminarverwaltung.de/seminare/?status=nd` und die Ressource der nicht durchführbaren Seminaranfragen wird repräsentiert. Darüber hinaus kann jede Repräsentation weitere Verweise zur Navigation beinhalten. Diese können wie in Listing 39 dargestellt, hinzugefügt werden.
XHTML setzt Outbound-Links durch das Verwenden von a- und link-Elementen um. Eingebettete Verweise können durch das src-Attribut kenntlich gemacht werden. Sie werden umgehend dereferenziert und die erhaltene Repräsentation wird in das Dokument eingebettet. XHTML unterstützt nur nicht-idempotente Methoden um auf eine Ressource zu schreiben. Formulare setzten den Hypermedia-Faktor Templated-Link um. So werden Vorlagen zur Verfügung gestellt, aus denen der Client einen gültigen Verweis erzeugen kann. Link-Annotation-Controls werden durch die Verwendung von Relationsbegriffen umgesetzt. Außerdem können Formulare zum Anlegen neuer Ressourcen genutzt werden. Das Formular fordert dann über input-Elemente die benötigten Werte an. Der Client kann auf diese Weise der Repräsentation entnehmen, wie eine neue Ressource zu erzeugen ist. Eine Erweiterung mit einem enctype-Attribut beschreibt den zu verwendenen Medientyp und entspricht dem Hypermedia-Faktor CU, weil der vom Dienstanbieter konsumierte Medientyp, reglementiert werden kann. Die Übermittlung an den Server erfolgt über die im method-Element festgelegte Methode und entspricht dem Hypermedia-Faktor CM. Listing 41 zeigt die umgesetzten H-Faktoren an einem Beispiel.

Listing 41: H-Faktoren XHTML

```
...
<link rel="Stornierungen"                                                <<CL
       href="http://seminaranfragen.de/storno"                           <<LO
       hreftype="application/xhtml+xml"></link>

<span class="stornierung" src="http://seminarverwaltung.de/storno/5512" <<LE
       srctype="application/xhtml+xml"></span>
```

```
<form name="neue-anfrage" method="post"                                    <<CM
       action="http://seminaranfragen.de/seminare"                         <<LT, LN
       enctype="application/json">                                         <<CU
  <input type="text" name="preis" />
  <input type="text" name="thema" />
  <input type="submit" value="Submit"/>
  </form>
...
```

Idempotente Methoden wie PUT oder DELETE können nach Spezifikation nicht verwendet werden. Control-Read wird ebenfalls nicht unterstützt, weil es für den Client nicht möglich ist, eine bevorzugte Repräsentation anzufordern. XHTML ist ein sehr allgemeines Format, welches durch die Verwendung von Relationsbegriffen an eine Problemdomäne angepasst werden kann. Die Steuerung der Anwendung kann über eingebettete Hypermedia-Elemente erfolgen. Tabelle 10 zeigt die Eigenschaften von XHTML im Überblick.

Tabelle 10: Überblick XHTML

H-Faktoren	H-Level	Design Elemente
LE, LO, LN, CU, CM, CL	Level 3	Base Format = XML, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich

4 Fazit und Ausblick

Der Einsatz von Hypermedia in einer verteilten Applikation hat diverse Vorteile gegenüber anderen, weit verbreiteten Ansätzen wie SOAP und RPC/RMI. Eine wesentliche Designentscheidung einer verteilten Anwendung ist es, wie private Datentypen und Ressourcen eines Dienstanbieters zugänglich gemacht werden können. SOAP, CORBA und RMI verwendet zu diesem Zweck Schnittstellen und Schnittstellenbeschreibungen. Schnittstellenbeschreibungen definieren das Zusammenspiel zwischen Client und Server auf einer sehr technischen Ebene. Diese Schnittstellenbeschreibungen sind oft sehr komplex und an eine konkrete Problemdomäne gekoppelt. Im Fall von SOAP und RMI muss der Client Kenntnis über die verwendeten Datentypen und die Rückgabewerte der aufgerufenen Methoden haben. Dieses Wissen muss im Client codiert werden und eine Änderung von den bestehenden Datentypen hat immer auch eine Änderung der bestehenden Dienstkonsumenten zur Folge.
Hypermedia verfolgt an dieser Stelle einen generischeren Ansatz. Die Kommunikation zwischen Client und Server wird auf einer höheren Ebene betrachtet, indem von Dingen der realen Welt abstrahiert wird. Im Umfeld einer Hypermedia-Anwendung existieren nur noch zwei wesentliche Dinge. Das sind zum einen Ressourcen und zum anderen Verweise die auf Ressourcen zeigen.
Selbstbeschreibende Nachrichten reichern die Kommunikation mit zusätzlicher Semantik an. Diese Semantik wir in der Regel über Relationsbegriffe umgesetzt. Sie beschreibt Beziehungen von Ressourcen untereinander, sowie zwischen einer Ressource und einem Client. Die Relationsbegriffe müssen dem Client bekannt sein und in einer Dokumentation festgelegt werden. Dieses Vorgehen beschränkt die notwendige Dokumentation im Idealfall auf die Relationsbegriffe. Einzelne Ressourcen mit den zur Verfügung stehenden Operationen müssen nicht erfasst werden, sondern sind dann Teil der selbstbeschreibenden Nachrichten und werden dem Client dynamisch zur Laufzeit übertragen. Auf diese Weise reduziert sich die Kopplung zwischen dem Client und seinem Dienstanbieter auf die verwendeten Relationsbegriffe. Dementsprechend sollte die Definition der Relationen möglichst stabil sein und keine hohe Änderungsfrequenz haben. Im Idealfall muss der Client einer Hypermedia-Anwendung dementsprechend nur Verweisen folgen und die verwendeten Relationsbegriffe kennen. Durch diese Entkopplung ergeben sich bei einer Hypermedia Anwendung eine Reihe von Vorteilen.
Da der Client, Verweisen entsprechen der ihm bekannten Relationen folgt, kann der Dienstanbieter seine Adressen ohne weiteres ändern. Nach der Änderung erhält der Client eine Repräsentation mit den neuen Adressen folgt ihnen auf die gleiche Weise wie er es mit den alten URIs getan hat. So erhält der Dienstanbieter eine hohe Flexibilität bezüglich des von ihm verwendeten Adressraumes. Die Verwendung von Hypermedia ist unabhängig von einem bestimmten Transportprotokoll. Die einzige Anforderung an das Protokoll ist es, dass es möglich sein muss Verweisen zu folgen. Somit kann auch das Protokoll entsprechend der Anforderungen gewählt und ausgetauscht werden. Die zwischen Client und Server ausgetauschten Repräsentationen sind unabhängig von den verwendeten Datentypen des Dienstanbieters. Der Server kann seine internen Datentypen ändern und erweitern ohne das die Kommunikation mit einem Client gestört wird, weil sich der

Nachrichtenaustausch auf die Repräsentationen beschränkt. Eine Hypermedia-Anwendung abstrahiert ebenfalls von dem zugrunde liegenden Geschäftsprozess. Der Client steuert die Anwendung über eingebettete Verweise entsprechend der bekannten Relationsbegriffe. Wenn der Geschäftsprozess verändert oder erweitert werden soll, kann der Dienstanbieter einfach neue Verweise hinzufügen, die der Dienstkonsument verwendet um die Applikation zu steuern. Bestehende Clients, die diese neuen Verweise und Relationen nicht kennen, funktionieren weiterhin, weil sie nur die ihnen bekannten Relationen verwenden. Das ermöglicht eine unkomplizierte Versionierung der Anwendung.[39]
Neben den genannten Vorteilen ergeben sich weitere, positive Eigenschaften, wenn HTTP als Transportprotokoll eingesetzt wird. HTTP ist ein sehr verbreitetes und erprobtes Protokoll. Eine Hypermedia-Anwendung kann von den in HTTP implementierten Mechanismen profitieren. HTTP stellt eine Reihe von Operationen zur Verfügung, die in der Applikation genutzt werden können. Methoden zum Lesen und Schreiben liegen in idempotenter und nicht idempotenter Form vor. Außerdem beinhaltet HTTP Mechanismen zur Datenkompression. Kompression kann in einer Hypermedia-Anwendung hervorragend eingesetzt werde, da in der Regel keine binären Formate eingesetzt werden und das aufkommende Datenvolumen mit bereits bestehenden Methoden erheblich verringert werden kann. Das Caching kann ebenfalls eingesetzt werden. Anwendungen, die eine große Anzahl von Clients bedienen, werden durch viele sich wiederholende Anfragen stark belastet. Diesem Problem kann das Caching von HTTP entgegenwirken. Betrachtet man beispielsweise den zentralen Einstiegspunkt einer Anwendung, wird dieser nur selten geändert aber von jedem Client angefragt. Im Allgemeinen kann das Caching von Ressourcen, die nur einer sehr geringen Änderungsfrequenz unterliegen, den Server entlasten und die Performance der Anwendung steigern. Zudem ist HTTP ein zustandsloses Protokoll und verfügt über sehr gute Skallierungseigenschaften.
Der wohl größte Nachteil einer Hypermedia-Anwendung ist die per Design erzeugte Latenz. Der Einsatz von Hypermedia eignet sich nicht für Anwendungen, die eine Kommunikation in Echtzeit erfordern. Das Serialisieren der Objekte in eine Repräsentation, sowie das Parsen und Deserialisieren erfordern Zeit. Die Verwendung von Caching-Strategien und Kompression kann diesen Effekt zwar mindern aber nicht vollständig beseitigen. Sollte eine Anwendung Kommunikation in Echtzeit erfordern, ist einem schnellen Kommunikationskanal in Verbindung mit binären Formaten der Vorzug gegenüber Hypermedia zu geben.
Der in dieser Arbeit erbrachte Vergleich verschiedener Formate betrachtet die grundlegenden Eigenschaften der Formate. Die Entscheidung, welches Format für einen bestimmten Anwendungsfall gewählt wird, hängt von den spezifischen Erfordernissen ab. Um einen Vergleich zu erbringen, wurden unterschiedliche Faktoren und Eigenschaften der Formate betrachtet. Tabelle 11 zeigt alle Formate mit ihren Eigenschaften im Überblick.

[39] Vgl. Amundsen, M. (2010), `http://amundsen.com/blog/archives/1069`

Tabelle 11: Überblick Hypermedia Formate

Format	H-Faktoren	H-Level	Design Elemente
Collection + JSON	LE, LO, LT, LI, LN, CL	Level 3	Base Format = JSON, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich
JSON-LD	LE, LO, LI, LN, CL	Level 3	Base Format = JSON, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich
JSON-Home	LE, LO, LT, LI, LN,CR, CU, CM, CL	Level 3	Base Format = JSON, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich
HAL	LE, LO, LT, LI, LN, CL	Level 3	Base Format = JSON, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich
Siren	LE, LO, LI, LN, CU, CM, CL	Level 3	Base Format = JSON, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich
Atom	LE, LO, LI, LN, CU, CL	Level 3	Base Format = XML, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich
XHTML	LE, LO, LN, CU, CM, CL	Level 3	Base Format = XML, State Transfer = AdHoc, Domain Style = allgemein, Application Flow = hauptsächlich

Grundsätzlich lassen sich alle Formate an eine spezifische Problemdomäne anpassen. Die hier betrachteten Medientypen verwenden als Basisformat JSON, XML oder bieten die Repräsentation in beiden Formaten an. Parser sind für beide Formate in vielen Sprachen implementiert und verfügbar. Eine generelle Empfehlung kann aufgrund der unterschiedlichen Anforderungen und benötigten Funktionen nicht gegeben werden. Die Unterstützung der einzelnen Hypermedia-Faktoren gibt jedoch Auskunft darüber, welche Eigenschaften ein Format nach Spezifikation bereits implementiert hat. Gegebenenfalls kann ein Format ausgewählt werden, dass die benötigten Eigenschaften bereits unterstützt.
Angenommen sei eine Anwendung zum Lesen verschiedener Nachrichtendienste. Diese Anwendung benötigt ausschließlich Funktionen die dem Lesen von Ressourcen dienen. Soll diese Anwendung erweitert werden, um auch Nachrichten zu verfassen ändern sich dementsprechend auch die Anforderungen an das verwendete Format. So sollte im Vorfeld geklärt werden, welche Anforderungen an das Format gestellt werden. Dazu zählen das Lesen und Schreiben von Ressource gleichermaßen wie die Möglichkeit Vorlagen oder idempotente Operationen zu verwenden.
Alle Formate unterstützen eine Hypermedia-Anwendung durch das Umsetzen verschiedener H-Faktoren. Ausgehend von den hier betrachteten Formaten bietet JSON-Home die umfangreichste Unterstützung von H-Faktoren und JSON-LD die am wenigsten ausgeprägte Unterstützung. Jedes Format setzt voraus, dass die genutzten Relationsbegriffe in einer externen Dokumentation erfasst werden. Der Umfang der Dokumentation hängt im Wesentlichen davon ab, ob die verschiedenen Control-Faktoren unterstützt werden. Sind Dinge wie die zu verwendende Operation oder der genutzte Datentyp, ein Teil der selbstbeschreibenden Nachricht kann auf diese Elemente der Dokumentation verzichtet werden. Andernfalls müssen die Ressourcen nicht nur semantisch beschrieben werden, sonder es müssen alle gültigen Operationen und Pflichtangaben erfasst werden.
Auch wenn es möglich ist, jedes Format um alle benötigten Eigenschaften zu erweitern, sollte dieser Erweiterung nicht dem Vorzug gegenüber einem Format gegeben werden, dass die benötigten Funktionen unterstützt. Eine starke Erweiterung der Spezifikation ist kontraproduktiv für die Interoperabilität des Formates. Generische, an ein Format angepasste Clients können theoretisch mit jeder Anwendung kommunizieren, die das Format verwenden. Diese Eigenschaft ist jedoch nur gegeben, wenn die Spezifikation nicht um weitere Hypermedia-Faktoren erweitert wurde. Eine weitere Herausforderung ist das Festlegen der Relationsbegriffe. Die Formate selbst definieren teilweise eine kleine Menge von Beziehungen, die entsprechend ihrer Spezifikation verwendet werden müssen. Für alle anderen benötigten Relationen, besteht die Möglichkeit die entweder selbst zu definieren oder auf eine verfügbare Definition zurück zugreifen.
Das Verwenden einer offiziell verfügbaren Definition hat den Vorteil, dass auch externe Entwickler diese einsehen können und die Entwicklung eines Clients für die Anwendung umsetzen können. Eine solche Menge von Definitionen stellt beispielsweise die Seite `http://schema.org/` zur Verfügung. Diese Seite definiert häufig genutzte Relationen wie Person, Kunde, Firma und stellen diese Definition öffentlich zur Verfügung. Die Definition beinhaltet den Namen der Relation und die zur Verfügung stehenden Eigenschaften. Ein Nachteil bei der Verwendung einer solchen Definition

besteht in dem zur Verfügung stehenden Gestaltungsfreiraum. So muss eventuell darauf verzichtet werden, eine Relation mit allen gewünschten Eigenschaften zu erfassen. Dafür bekommt man eine vorgefertigte Spezifikation von Dingen der realen Welt.
Hypermedia löst durch Abstraktion, an vielen Stellen, Probleme die bei der Erstellung eines verteilten Informationssystems auftreten. Der Grad der Kopplung zwischen Client und Server wird minimiert und die Dokumentation kann auf einer sehr hohen Ebene geschrieben werden. Im wesentlichen muss sie die verwendeten Relationsbegriffe beinhalten. Komplexe Schnittstellenbeschreibungen sind nicht notwendig, da ein Dienstkonsument die Anwendung durch eingebettete Verweise und Vorlagen steuern kann. Die Navigation durch den Anwendungsprozess erfolgt ebenfalls durch eingebettete Verweise. Die Entkopplung sorgt dafür, dass Hypermedia-Anwendungen eine gute Wartbarkeit und Erweiterungsmöglichkeit besitzen. Der Nachteil, der prinzipbedingt auftretenen Verzögerung sollte für die meisten wirtschaftlichen Unternehmensanwendungen nicht zum tragen kommen, da ein Benutzer eine Verzögerung in diesem Rahmen nicht merkt.
Der Einsatz von Hypermedia hilft Probleme bei der Erstellung einer verteilten Anwendung zu lösen und ist ein solides Konzept um zukünftige Anwendung robust, erweiterbar und skallierbar zu entwerfen.

*Literaturverzeichnis

Amundsen, M. (2010a). *Hypermedia Levels, http://amundsen.com/hypermedia/scraps/hypermedia-levels, 09.08.13*.

Amundsen, M. (2010b). *the advantages of hypermedia types, http://amundsen.com/blog/archives/1069, 09.08.13*.

Amundsen, M. (2012). *Building Hypermedia APIs with HTML5 and Node*. OReilly.

Amundsen, M. (2013). *Collection+JSON - Document Format, http://amundsen.com/media-types/collection/format/, 09.08.13*.

Berners-Lee, T. (1990). *WorldWideWeb: Proposal for a HyperText Project, http://www.w3.org/Proposal, 09.08.13*.

Birbeck, M. (2010). *XHTML 2.0, http://www.w3.org/TR/xhtml2/xhtml2.html, 09.08.13*.

Crockford, D. (2006). *The application/json Media Type for JavaScript Object Notation (JSON), http://www.ietf.org/rfc/rfc4627, 09.08.13*.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. UNIVERSITY OF CALIFORNIA, IRVINE.

Gregorio, J. (2007). *The Atom Publishing Protocol, http://tools.ietf.org/html/rfc5023, 09.08.13*.

Kelly, M. (2013). *JSON Hypertext Application Language, http://tools.ietf.org/html/draft-kelly-json-hal-05, 09.08.13*.

Nottingham, M. (2005). *The Atom Syndication Format, http://tools.ietf.org/html/rfc4287, 09.08.13*.

Nottingham, M. (2013). *Home Documents for HTTP APIs, http://tools.ietf.org/html/draft-nottingham-json-home-03, 09.08.13*.

Richardson, L. (2008). *The Maturity Heuristic, http://www.crummy.com/writing/speaking/2008-QCon/act3.html, 09.08.13*.

Sporny, M. u. (2013). *JSON-LD 1.0 JSON-LD 1.0 - A JSON-based Serialization for Linked Data, http://json-ld.org/spec/latest/json-ld/, 09.08.13*.

Swiber, K. (2013). *Siren: a hypermedia specification for representing entities, https://github.com/kevinswiber/siren, 09.08.13*.

Ehrenwörtliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Bachelorthesis selbstständig und ohne unerlaubte fremde Hilfe angefertigt habe, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Meinersen, den 12. August 2013