2.3 Standardisierung mit dem Open Data Protocol

Das Open Data Protcol (OData) ist seit dem 23. Februar 2017 ein offiziell anerkannter ISO und IEC Standard [CG17]. Ursprünglich wurde OData von Microsoft entwickelt. Dies hat den positiven Nebeneffekt, dass die meisten Microsoft Produkte diesen Standard unterstützen. Der Standard wurde anschließend an OASIS übergeben, um auf Open Source Basis die Entwicklung eines einheitlichen Standards für REST Schnittstellen voranzubringen. Die aktuelle Version 4.0 des Protokolls wurde von OASIS entwickelt. Das Ziel dieses Protokolls ist es, einen einheitlichen Standard für eine Client-Server Kommunikation zu etablieren. Der Fokus liegt also auf dem Austausch von Daten zwischen verschiedenen Systemen. OData bezeichnet sich selbst, als „the best way to REST“ und stellt einen Standard zur Verfügung, um die Prinzipien und Architektur einer RESTful API umzusetzen.

Bei dem Standard spielen verschiedene Begriffe wie Entitäten, Metadaten und Query Operators eine zentrale Rolle. Dabei baut OData auf die bereits bestehenden technischen Möglichkeiten HTTP, Atom Publishing Protocol und JSON auf [Nas16].

2.3.1 Entity Data Model

Die Basis von OData ist das Entity Data Model. Dieses ist ein abstraktes Datenmodell, dass die Daten beschreibt, die von der Schnittstelle zur Verfügung gestellt werden [Oda, Part 1, Kapitel 3]. Das Entity Data Model umfasst folgende Elemente [Olia, Entity Data Model (Metadata)]:

- Schemas
- Entity Types
- Complex Types
- Enum Types
- Properties
- Navigation Properties
- Actions
- Functions
- Entity Container

Ein Kernkonzept dieses Datenmodells sind Entitäten, die mit Hilfe von Metadaten beschrieben werden. Eine Entität ist eine definierte Klasse mit bestimmten Eigenschaften [Abt]. Die Beschreibungen des Entity Data Models, beziehungsweise die Entitäten, die von einem Service zur Verfügung gestellt werden, werden in einem Metadaten-Dokument festgehalten, das vom Client abgerufen werden kann. Des Weiteren muss ein Service Dokument zur Verfügung gestellt werden, das die Einstiegspunkte und spezielle Operationen listet [Bol14b]. Der größte Vorteil dieser Metadaten ist, dass sich ein Client daraus automatisch Klassen generieren kann, die die entsprechenden Entitäten abbilden. Da dies programmiersprachenunabhängig geschieht, kann jeder

Entwickler, egal welche Sprache dieser präferiert, die von der Schnittstelle zur Verfügung gestellten Daten einfach abrufen und nutzen [Abt]. Beide Dokumente müssen über eine bestimmte URL aufrufbar sein:

Beispiel:

http://uni-bamberg/service/$metadata

Aufruf des Metadatendokuments über die URL des Services

Ein Beispiel für eine Entität ist im Hochschulkontext die Klasse Lehrveranstaltung. Bei mehreren Entitäten spricht man auch von einer Collection oder einem Entity Set [Bol14b]. Diese Entitäten können verschiedene *Properties* besitzen, wie zum Beispiel einen Namen, eine Beschreibung oder ein Datum. Zusätzlich zu den Properties ist eine Referenzierung auf andere Entitäten mittels *Navigation Properties* möglich. Eine mögliche Entität, auf die das *Navigation Property* einer Lehrveranstaltung verweist, ist ein Dozierender. Die Dozierendenentität besitzt wiederum eigene Properties, wie Name, E-Mail Adresse und Anschrift [Bol14b]. Auf die verschiedenen Typen wird im Abschnitt „Primitive Types, Complex Types und Enumeration Types“ näher eingegangen. Beim Abruf der Daten kann der Client zwischen einzelnen Entitäten oder einer gesamten Collection beziehungsweise Entity Set wählen. Alle Entity Sets und Entitäten werden in einem Entity Container dem Client zur Verfügung gestellt [Oda, Part 1, Kapitel 10.3].

Beispiel:

http://uni-bamberg/service/Courses(1)

Aufruf der Entität Lehrveranstaltung mit der ID 1

http://uni-bamberg/service/Courses

Aufruf der Collection, die alle Lehrveranstaltungsentitäten enthält

http://uni-bamberg/service/Courses(1)/Performer

Aufruf der Collection Performer, die von der Lehrveransatltungsentität mit der ID 1 referenziert werden

2.3.2 Primitve Types, Complex Types und Enumeration Types

Wie bereits erwähnt, kann eine Entität verschiedene Properties besitzen. Diese können in Primitive Types, Complex Types und Enumeration Types unterschieden werden. Alle drei Types können der übergeordneten Gruppe der Structured Types zugeordnet werden [Oda, Part 3, Kapitel 4.2 und 4.3]. Primitive Types sind Property Typen, die von OData zur Verfügung gestellt werden. Dies umfasst beispielsweise die Typen Binary, Boolean, String, Date. Primitive Types werden also für einfache Properties genutzt, die den Standard Datentypen entsprechen. Eine Auflistung aller zur Verfügung gestellten Primitive Types findet sich in der OData Dokumentation[1].

[1] OData Dokumentation, Primitive Types, http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/complete/part3-csdl/odata-v4.0-errata03-os-part3-csdl-complete.html#_Structured_Types

Complex Types sind schlüssellose, nominale, strukturierte Typen. Nominal bedeutet, dass der Typ einen eindeutigen Namen haben muss, über den die entsprechende Ressource referenziert werden kann. Dies kann über einen sogenannten „Namespace-qualified name“ oder über einen „Alias-qualified name“ geschehen [Oda, Part 3, Kapitel 9].

Beispiel:

```
<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
        Namespace="org.example"
        Alias="einrichtung">
  <ComplexType Name="Addresse">...</ComplexType>
</Schema>
```

Der nominale Type kann demnach über den Namen ...org.example.Adresse oder über den Alias ...einrichtung.Adresse aufgerufen werden.

Das ein Complex Type schlüssellos ist bedeutet, dass er nicht unabhängig von einem Entity Type referenziert, erstellt oder aktualisiert werden kann. Beispielsweise wäre ein Aufruf des Complex Types Raum nur über „http://uni-bamberg.service/Lehrveranstaltung(1)/Raum(1)“ möglich, nicht aber über „http://uni-bamberg.service/Raum(1)“, da eine Referenzierung ohne Angabe des Entity Types nicht möglich ist [Oda, Part 3, Kapitel 9].

Des Weiteren kann ein Complex Type in zwei Arten unterschieden werden: in *Structural Property* und *Navigation Property*. Für beide Ausprägungen des Complex Types gilt, dass alle Properties innerhalb eines Complex Types einen eindeutigen Namen haben müssen [Oda, Part 3, Kapitel 9]. Ein Structural Property ist eine benannte Referenz zu einem Type oder einer Collection, die entweder primitiv, komplex oder ein Enum ist. Die benannte Referenz zeigt demnach auf einen weiteren Structured Type [Oda, Part 3, Kapitel 6].

Ein Navigation Property hingegen ist eine benannte Referenz zu einer Entität oder Collection. Das Navigation Property kann entweder Teil eines Complex Types sein oder außerhalb eines Complex Types der Entität zugeordnet sein. Wenn das Navigation Property Teil eines Complex Types ist, kann dieses nicht unabhängig von der eigentlichen Entität geändert oder referenziert werden [Oda, Part 3, Kapitel 7]. Das Navigation Property kann also in beiden Fällen auf eine Entität verweisen, die wiederum verschiedene Properties halten kann. Der Unterschied liegt nur in der Adressierung und bei den damit einhergehenden Regeln zur Bearbeitung und Referenzierung. Im Kapitel „Umsezung“, wird veranschaulicht, wie eine Entität ein NavigationProperty besitzt, dass auf einen bestimmtem Typ von Entität oder Collection verweist. Beispielweise verweist eine Lehrveranstaltungsentität mit einem NavigationProperty auf Dozierende. Die logische Zuweisung auf welche Dozierenden von einer Lehrveranstaltung aus verwiesen wird, bleibt dem Entwickler überlassen und wird nicht vom eigentlichen OData Protokoll gesteuert.

2.3.3 Query Operators

Ein weiterer Kernpunkt von OData ist die Nutzung von Query Operators. Dem Client werden verschiedene Query Options zur Verfügung gestellt mit denen eine Eingrenzung der Anfrage von Entitäten ermöglicht wird. Mögliche Query Options sind [Oda, Part 1, Kapitel 11.2]:

- $search
- $filter
- $count
- $orderby
- $skip
- $top
- $expand
- $select
- $format

Durch die System Query Options lassen sich eine Vielzahl von komplexen Anfragen ausführen. Der Client kann direkt in der URL spezifizieren, welche Entitäten er genau benötigt. Dadurch müssen nicht alle Daten aus einer Datenbank geladen werden, sondern nur die, die vom Client benötigt werden. Die unterschiedlichen Query Options und deren Funktion werden im folgenden Abschnitt näher erläutert.

$count

Die Query Option $count zählt die Gesamtzahl der Entitäten, die durch den Request des Client abgerufen wurden. Diese wird dem zurückgegeben Ergebnis als Attribut *count* hinzugefügt. Um die Count Option zu aktivieren, muss der Client diese in der aufgerufenen URI als „true“ deklarieren. Des Weiteren bezieht sich $count immer auf die Gesamtzahl der Entitäten, unabhängig davon, ob das Ergebnis von den Filteroptionen $top, $skip oder $expand verändert wurde [Oda, Part 1, Kapitel 11.2.5.5][Olif, Kapitel 3.1].

Beispiel und Ergebnis:

http://uni-bamberg/service/Courses?$count=true

Aufruf der Collection Courses mit Query Option count

```
1 {"@odata.context":"$metadata#Courses",
2 "@odata.count":3,
3 "value":[
4         {"ID":1,
5         "Name":"Testkurs1",
6         "Description":"Der erste TestKurs",
```

```
7           "MaximumParticipants":20},
8           {"ID":2,
9           "Name":"Testkurs2",
10          "Description":"Der zweite TestKurs",
11          "MaximumParticipants":5},
12          {"ID":3,
13          "Name":"Testkurs3",
14          "Description":"Der dritte TestKurs",
15          "MaximumParticipants":120}]}
```

$orderby

Mit der Query Option $orderby ist es möglich die von dem Service abgefragten Daten zu ordnen. Der Client kann entweder ein Attribut oder mehrere Attribute angeben, nach dem die Entitäten geordnet werden. Des Weiteren kann der Client die Typen *asc* und *desc* angeben, um zu bestimmen, ob die Sortierung auf- oder absteigend erfolgen soll. *Asc* steht für ascending, also aufsteigend und *desc* für descending, übersetzt absteigend. Wenn kein Typ angegeben ist, erfolgt die Sortierung automatisch aufsteigend. Wie die eigentliche Sortierung der Entitäten abläuft ist nicht spezifiziert und wird durch die Implementierung des Entwicklers bestimmt. Im Kapitel „Umsetzung“ wird dieser Aspekt wieder aufgegriffen [Oda, Part 1, Kapitel 11.2.5.2][Olih, Kapitel 3.1].

Beispiel:

http://uni-bamberg/service/Courses?$orderby=MaximumParticipants asc, Name desc

Zunächst Aufruf der Collection Courses mit aufsteigender Sortierung nach Anzahl der maximalen Teilnehmer; bei gleicher Anzahl erfolgt eine absteigende Sortierung anhand des Namens

$skip und $top

Durch die Query Option $top kann der Client spezifizieren, wie viele Entitäten er maximal als Rückgabewert haben möchte. Die Collection wird dabei auf den angegebene Wert reduziert. Bei einer Collection die 10 Entitäten umfasst, werden bei einem Wert von „4“nur die ersten vier Entitäten in der Collection dem Client zurückgegeben [Oda, Part 1, Kapitel 11.2.5.3][Olif, Kapitel 3.3]. Die Query Option $skip kann als Gegenstück zu $top gesehen werden. Mit dieser Option kann der Client bestimmen, an welcher Stelle die Collection, beziehungsweise Liste von Entitäten starten soll. Alle Entitäten vor dieser Stelle werden ignoriert. Bei einem angegeben Wert n startet die Collection an der Stelle n+1 [Oda, Part 1, Kapitel 11.2.5.4][Olif, Kapitel 3.2].

Beispiel:

http://uni-bamberg/service/Courses?$top=3

Aufruf der Collection Courses mit maximal drei Entitäten

http://uni-bamberg/service/Courses?$skip=5

Aufruf der Collection Courses, startend bei Eintrag sechs der Collection

$select

Bei der Anfrage einer Collection oder Entität kann der Client mit der Query Option $select den Umfang der zurückgegebenen Daten des Services einschränken. Der Client benötigt beispielsweise nicht alle Properties einer Lehrveranstaltungsentität, sondern nur die Anzahl der maximalen Teilnehmer und legt das benötigte Property mittels $select fest. Es besteht die Möglichkeit nur einzelne Properties, mehrere Properties oder alle Properties abzufragen. Um alle Properties abzurufen, kann der Client „*“ anstelle von Properties angeben [Oda, Part 1, Kapitel 11.2.4.1][Olig, Kapitel 3.1].

Beispiel:

http://uni-bamberg/service/Courses?$select=MaximumParticipants, Description

Aufruf der Collection Courses, die als Rückgabewert nur die Properties MaximumParticipants und Description besitzen

*http://uni-bamberg/service/Courses?$select=**

Aufruf der Collection Courses mit allen Properties

$search

Mit $search kann der Client nach einem spezifizierten String suchen. Dabei können verschiedene Strings auch mit AND oder OR kombiniert werden. Auch eine Einschränkung der Suchergebnisse ist mit NOT möglich. Nach welchem Attribut gesucht wird, beziehungsweise wie die Suche abläuft, obliegt der Implementierung des Programmierers [Oda, Part 1, Kapitel 11.2.5.6].

Beispiel:

http://uni-bamberg/service/Courses?$search=algorithmen AND NOT medieninformatik

Aufruf aller Lehrveranstaltungsentitäten die das Wort „algorithmen“ und nicht „medieninformatik“ enthalten.

$filter

Mit $filter lässt sich ein bestimmtes Attribut der Entität nach spezifizierten Kriterien filtern. Dafür stehen dem Client Vergleichsoperatoren, Logikoperatoren und Arithmetikoperatoren zur Verfügung. Allgemein lassen sich unterschiedliche Query Options immer durch das Symbol „&“ kombinieren. Die Kombination der Query Options $filter und $top führt beispielsweise dazu das gefilterte Ergebnis auf eine bestimmte Anzahl von Entitäten zu limitieren [Oda, Part 1, Kapitel 11.2.5.1].

Beispiel:

http://uni-bamberg/service/Courses?$filter=Credits gt 3

Aufruf von Lehrveranstaltungen, die mehr als 3 Credits geben

http://uni-bamberg/service/Courses?$filter=Credits gt 3&$top=5

Aufruf der ersten 5 Lehrveranstaltungen, die mehr als 3 Credits geben

Des Weiteren unterscheidet OData die Filtermöglichkeiten in zwei Arten: *Built-in Filter Operations* und *Built-in Query Functions*. Die zur Verfügung stehenden *Filter Operations* sind in folgender Tabelle aufgelistet:

Operator	Beschreibung
eq	gleich
ne	nicht gleich
gt	größer als
ge	größer als oder gleich
lt	kleiner als
le	kleiner als oder gleich
has	hat Kennzeichen
and	logisches und
or	logisches oder
not	logische Verneinung
add	Addition
sub	Subtraktion
mul	Multiplikation
div	Division
mod	Modulo
( )	Gruppierung

Tabelle 2.2 — Überblick Filter Operatoren [Oda, Part 1, Kapitel 11.2.5.1.1]

Die *Query Functions* ermöglichen das Bilden komplexerer Filtermöglichkeiten. Die bereitgestellten Funktionen ermöglichen die Filterung der Inhalte von Properties, die zum Beispiel ein String oder ein Datum sind. Eine Liste aller möglichen *Query Functions* findet sich in der OData Dokumentation[2]

Beispiel:

http://uni-bamberg/service/Courses?$filter=contains(Description, 'Algorithmen')

[2]Part 1, 11.2.5.1.2 Built-in Query Functions, http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html

Aufruf von Lehrveranstaltungen, die im String-Attribut „Decription“ das Wort „Algorithmen“ enthalten.

http://uni-bamberg/service/Courses?$filter=length(Description) lt 100

Aufruf von Lehrveranstaltungen, bei denen das String-Attribut „Description“ weniger als 100 Zeichen besitzt

http://uni-bamberg/service/Courses?$filter=date(CourseStartTime) ne date(CourseEndTime)

Aufruf von Lehrveranstaltungen, bei denen der Starttermin des Kurses nicht dem Endtermin entspricht

$expand

Durch die Filter Option $expand wird das Abrufen eines Datensatz mit verknüpften Entitäten ermöglicht. Eine Entität und deren über Navigation Properties verknüpfte Entitäten kann somit durch eine einzelne Abfrage abgerufen werden. In allen bisher gezeigten Beispielen wurde eine Entität oder Collection eines bestimmten Types abgerufen. Bisher bezogen sich die Beispiele auf den Typ „Lehrveranstaltung“ und „Dozierender“. Bekanntermaßen läuft die Referenzierung von einer Entität zu einer anderen Entität über ein Navigation Property ab. Bei dem Abruf von Lehrveranstaltungen möchte der Client jedoch gegebenenfalls die Daten einer Lehrveranstaltung mit denen der dazugehörigen Dozierenden gleichzeitig abrufen, ohne zwei getrennte Abfragen auszuführen. Dies wird durch $expand ermöglicht. Der Client kann hierbei angeben welche NavigationProperties der Entität geladen und dem Datensatz hinzugefügt werden sollen. Dies hat den Vorteil, dass nur die Daten geladen werden müssen, die auch benötigt werden. [Oda, Part 1, Kapitel 11.2.4.2][Olig, Kapitel 3.2].

Beispiel und Ergebnis:

http://uni-bamberg/service/Courses(1)?$expand=Performer

Aufruf der ersten Course Entität mit expandierten NavigationProperty Performer

```
1  {"@odata.context":"$metadata#Courses/$entity",
2  "ID":1,"Name":"Testkurs1",
3  "Description":"Der erste TestKurs",
4  "MaximumParticipants":20,
5  "Performers":[
6          {"ID":1,
7          "Name":"Jonathan Boss",
8          "EMail":"jonathan.boss@uni-bamberg.de"},
9          {"ID":2,
10         "Name":"Martin Mai",
11         "EMail":"martin.mai@uni-bamberg.de"}]}
```

$format

Mittels der System Query Option $format kann der Client das Format der Antwort des Services festlegen. Ein OData Service unterstützt das OData-Atom Format und das Odata-JSON Format. Sollte der Client kein Format angeben, erfolgt die Ausgabe standardmäßig im JSON Format [Oda, Part 1, Kapitel 11.2.10].

Beispiel:

http://uni-bamberg/service/Courses?$format=application/json

Aufruf der Collection Courses im JSON Format

2.3.4 Create, Update, Delete

Eine REST Schnittstelle, die dem OData Protokoll entspricht, unterstützt nicht nur Leseoperatoren sondern auch Schreiboperationen. Vorwiegend schließt dies das Anlegen, Verändern und Löschen von Entitäten ein [Oda, Part 1, Kapitel 11.4]. Da die umzusetzende Schnittstelle mit Lehrveranstaltungsdaten nur Lese-Operationen erfordert, wird diese Thematik nur begrenzt behandelt.

Mittels eines HTML POST Request lässt sich eine neue Entität erstellen. Der Request Body muss der Repräsentation der Entität entsprechen. Wenn die anzulegende Entität eine Verknüpfung zu einer anderen Entität benötigt, muss die dazugehörige Entität über die ID referenziert werden und im Request Body mit angegeben werden. [Oda, Part 1, Kapitel 11.4.2]. Ein möglicher Request Body zum Anlegen einer Lehrveranstaltung mit den Attributen ID, Name, Description und MaximumParticipants würde folgendermaßen aussehen:

```
1 {"ID":4,
2 "Name":"Testkurs4",
3 "Description":"Der vierte TestKurs",
4 "MaximumParticipants":420 }
```

Um eine Entität zu verändern stehen, die HTML Methoden PATCH und PUT zur Verfügung. Jedoch wird die Operator PATCH als Methode für eine OData Schnittstelle empfohlen [Oda, Part 1, Kapitel 11.4.3]. Bei einem PATCH Request werden nur die mitgeschickten Properties der Entität geändert. Bei einem PUT Request werden sämtliche Properties überschrieben. Ausgenommen sind davon Schlüsselattribute [Olid, Kapitel 3.1, 3.2]. Wenn Properties nicht mitgeschickt wurden, werden sie auf *null* gesetzt. Eine Unterscheidung der beiden Methoden muss demnach auch in der Implementierung erfolgen. Mit dem folgenden Request Body würden bei PATCH nur die angegebenen Properties „Name“ und „MaximumParticipants“ geändert werden. Demgegenüber würden bei PUT die „Description“ auf *null* gesetzt werden, da diese nicht mit angegeben ist.

```
1 {
2 "Name":"Testkurs der Dritte",
3 "MaximumParticipants":20 }
```

Bei der DELETE Operation wird die entsprechende Entität gelöscht. Der Aufruf erfolgt mit einem leeren Request Body [Oda, Part 1, Kapitel 11.4.5].

2.3.5 Operations

In der OData Dokumentation findet sich folgende Definition zu Operations: „Operations allow the execution of custom logic on parts of a data model. Functions are operations that do not have side effects and may support further composition, for example, with additional filter operations, functions or an action. Actions are operations that allow side effects, such as data modification, and cannot be further composed in order to avoid non-deterministic behavior.“ [Oda, Part 1, Kapitel 3].

Operationen lassen sich demnach in *Functions* und *Actions* unterteilen. Eine Operation ermöglicht die Ausführung von programmierter Logik. Eine *Function* unterscheidet sich primär von einer *Action*, indem sie keine Seiteneffekte erzeugen können. Im Gegensatz zur *Function* kann eine *Action* folglich auch die Daten einer Schnittstelle modifizieren.

Das Ergebnis einer solchen Operation kann folgenden Datentypen entsprechen [Olij, Introduction]:

- Einer Entität oder einer Collection von Entitäten
- Einer Primitive Property oder eine Collection von Primitive Properties
- Einer Complex Property oder einer Collection von Complex Properties
- void (nur bei einer Action)

2.3.6 Apache Olingo

Apache Olingo ist eine Java Bibliothek, die die Standards von OData implementiert. Olingo stellt dabei verschiedene Funktionen wie URL Parsing, Input Validierung, Serialisierung von Inhalten, Anfragenverteilung gemäß der OData Spezifikation zur Verfügung“ [Olia]. Der Entwickler muss sich somit nicht mehr um die Spezifikation und Kommunikation eines solchen Services kümmern, sondern kann den Fokus auf die Anwendungslogik und die Daten der Anwendung legen [Bol14a].

Aufgrund der Tatsache, dass OData sprachenunabhängig ist, gibt es eine Vielzahl verschiedener Bibliotheken und Tools. Da die praktische Umsetzung der Schnittstelle in dieser Bachelorarbeit in Java erfolgt, ist Apache Olingo die zu nutzende Bibliothek für die spätere Implementierung der Schnittstelle.