Grenzenlos mobil

Plattformübergreifende Datenintegration mit dem Open Data Protocol

Klaus Rohe

Das Open Data Protocol, kurz OData genannt, ist eine Spezifikation von Microsoft, welche ein Protokoll für die plattformübergreifende Integration von Daten definiert. OData basiert auf REST und den Formaten AtomPub sowie JSON. Außer für Microsoft .NET 3.5SP1 und .NET 4.0 gibt es unter anderem SDKs für Java, JavaScript, PHP und ObjectiveC. IBM stellt bereits für die „WebSphere eXtreme Scale"-Plattform einen zu OData kompatiblen Daten-Service bereit. Der Artikel gibt eine Einführung in OData und zeigt anhand von existierenden Anwendungen das Einsatzspektrum dieser Technologie auf.

Hintergrund

Der Ursprung von OData ist ein Projekt bei Microsoft mit dem Namen Astoria. Astoria wurde im Oktober 2008 erstmals öffentlich vorgestellt. Die Idee dahinter ist, Daten im AtomPub- (Atom Publishing Protocol) oder JSON-Format (JavaScript Object Notation) für den lesenden und schreibenden Zugriff als REST-Webservices bereitzustellen. Astoria wurde dann als „ADO.NET Data Services" in das .NET Framework 3.51 aufgenommen und ist in .NET 4.0 unter dem Begriff „WCF Data Services" zu finden.

Im Oktober 2009 hat Microsoft auf der Professional Developers Conference in Los Angeles bekannt gegeben, dass die Spezifikationen des Protokolls und der Datenformate, welche die Basis der WCF Data Services sind, allgemein zur Verfügung gestellt werden. Die Spezifikationen sind unter dem in [OData] aufgeführten Link zu finden, ebenso die Links zu den OData SDKs für Java, ObjectiveC, PHP und andere Programmiersprachen. Einen guten Einstieg in die Thematik bieten auch die Fragen und Antworten zu OData auf [ODataQA].

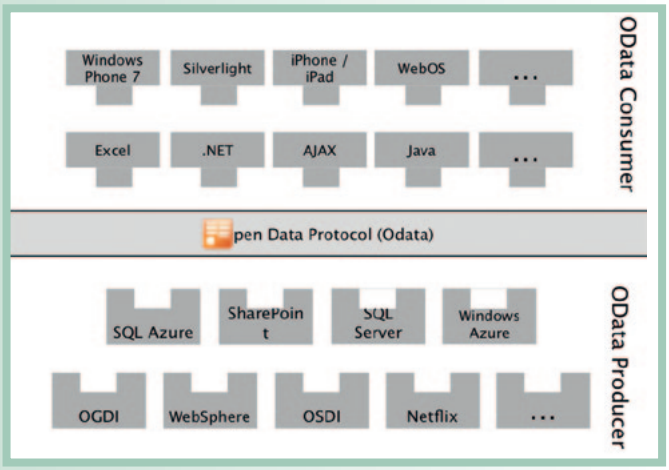

Abb. 1: Einsatzszenarien von OData-Clients (Consumer) und -Servern (Producer)

Abbildung 1 zeigt die heute möglichen Einsatzszenarien von OData. Wie zu ersehen ist, können Applikationen, welche die Rolle eines OData-Consumers spielen, für alle gängigen Client-Betriebssysteme, Browser, Office-Applikationen und Smart-Phones entwickelt werden.

OData hat das Potenzial, in Zukunft für Web- und Cloud-Applikationen eine analoge Rolle zu spielen, wie ODBC für Client-Server-Applikationen in den 1990-Jahren. Obwohl OData seinen Ursprung in den „ADO.NET Data Services" des Microsoft .NET Frameworks hat, ist es nicht auf relationale Daten beschränkt, sondern es wird ein abstraktes Datenmodell aus Entitäten und deren Assoziationen benutzt. OData ist daher nicht an bestimmte Datenbanktypen oder Persistenzmechanismen gebunden.

Im Folgenden wird die Struktur von OData an einem Beispiel-Service dargestellt, dessen URI auf der Web-Seite [OData] zu finden ist und der dort zur Verfügung gestellt wird.

Struktur von OData

Jeder OData-Service veröffentlicht ein *Service Document* und optional ein *Service MetadataDocument*, das die Daten des Service mit einem abstrakten Datenmodell beschreibt, welches als „Entity Data Model" (EDM) bezeichnet wird. EDM hat sehr große Ähnlichkeit mit dem Entity-Relationship-Modell von Chen [WikiERM].

Service-Dokumente, URI-Konventionen und Operationen

Das „Service Document" ist der Einstiegspunkt in einen OData-Service. Es liefert eine Liste aller Datenkategorien, die ein OData-Service bereitstellt, im AtomPub-Format [Til09]. Der Einstiegspunkt des Service wird über einen URI angesprochen, z. B. `http://services.odata.org/OData/OData.svc/`. Dies ist der oben erwähnte Beispiel-Service, den man für Tests und eigene Experimente mit OData nutzen kann.

Das Datenmodell, welches sich hinter dem Service verbirgt, wird durch das in Abbildung 2 dargestellte EDM beschrieben. Spricht man den oben genannten URI z. B. über den Web-Browser an, so bekommt man das in Abbildung 3 dargestellte XML-Dokument im AtomPub-Format zurück.

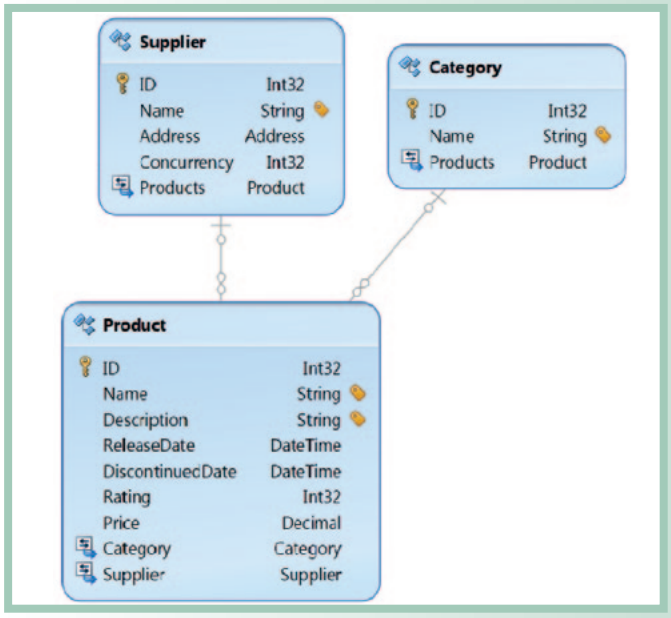

Abb. 2: EDM des Service http://services.odata.org/OData/OData.svc/

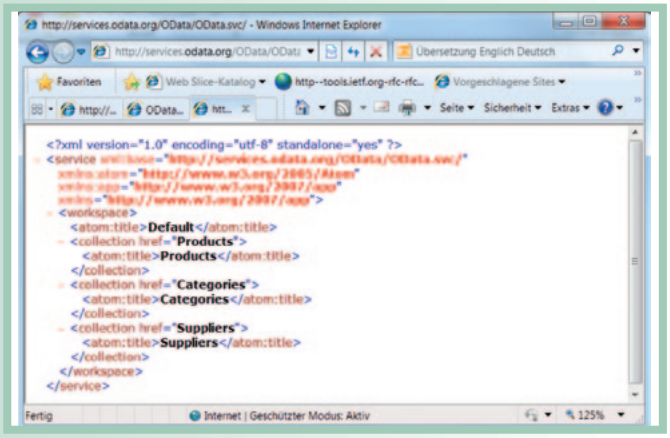

Abb. 3: Ergebnis einer Browser-Abfrage des URI `http://services.odata.org/OData/OData.svc/`

Die Daten, auf die man über diesen Service zugreifen kann, sind also die „Collections“, „Products“, „Categories“ und „Suppliers“ (s. a. Abb. 2). Will man z. B. auf alle Produkte lesend zugreifen, so muss ein HTTP GET-Request mit dem folgenden URI erzeugt werden: `http://services.odata.org/OData/OData.svc/Products`. Will man nur das Produkt mit dem Primärschlüssel 3 (ID in der Entität Products in Abb. 2) ansprechen, so wird dies mit dem folgenden URI erreicht: `http://services.odata.org/OData/OData.svc/Products(3)`.

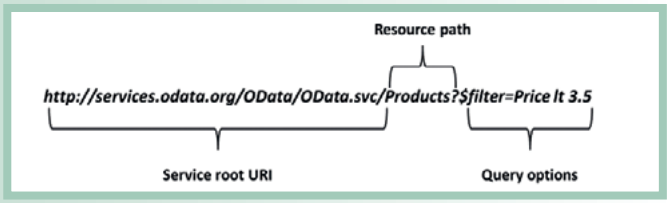

Abb. 4: Komponenten eines URI für OData

Der Aufbau eines URI für OData ist exemplarisch in Abbildung 3 dargestellt. Die drei Komponenten sind

- die „Service root URI“, also der Einstiegspunkt in den Service,
- der „Resource path“, welcher die Ressource spezifiziert, auf die eine Operation angewandt werden soll, und
- als dritte Komponente die „Query options“, in der Filterbedingen usw. definiert werden können.

Ein weiteres Beispiel für einen OData URI ist `http://services.odata.org/OData/OData.svc/Categories(1)/Products`. Dieser URI identifiziert alle Produkte der Kategorie mit ID=1, „/Categories(1)/Products“ entspricht hier dem „Resource path“. Die Anzahl der Produkte in der Kategorie mit ID=1 bekommt man mit dem URI `http://services.odata.org/OData/OData.svc/Categories(1)/Products/$count`.

Das Repräsentationsformat der Daten ist bei OData auf AtomPub eingestellt. Will man als Repräsentationsformat JSON, so muss dies in dem URI spezifiziert werden, beispielsweise `http://services.odata.org/OData/OData.svc/Categories(1)/Products?$format=json`. Dies wird also mit der „Query option“ `$format=json` erreicht. Die kompletten Details zum Aufbau von OData URIs findet man in [OData] im Kapitel „Protocols“ im Abschnitt „URI Conventions“.

Generell betrachtet stellt OData eine einheitliche Schnittstelle dar, um Operationen auf strukturierten und unstrukturierten Daten durchzuführen. Der Großteil der Semantik der von OData unterstützten Operationen wurde vom AtomPub-Protokoll übernommen, welches selbst wieder auf HTTP aufsetzt.

OData stellt die Operationen Create, Retrieve, Update und Delete, kurz CRUD genannt, zur Verfügung, welche auf die HTTP-Standardverben POST, GET, PUT (MERGE) und DELETE abgebildet werden. Der Unterschied von PUT und MERGE im Rahmen von OData besteht darin, dass MERGE dazu benutzt wird, nur einzelne Felder bzw. Eigenschaften zu verändern. Beispielsweise, wenn man in „Product“ (s. Abb. 2) nur den Preis verändern will, aber alle anderen Felder in ihrem alten Zustand lässt.

Durch die Nutzung von MERGE, welches kein HTTP-Standardverb ist, vermeidet man ein Überladen von PUT. Es wird aktuell darüber diskutiert, ob das Verb PATCH in die offizielle HTTP-Spezifikation übernommen werden soll. Sobald dies geschieht, wird OData das Verb PATCH einführen und die Unterstützung von MERGE auslaufen lassen.

Die Einzelheiten zu den von OData unterstützten Operationen findet man in [OData], Kapitel „Protocols“, im Abschnitt „Operations“.

Metadaten-Dokumente

Ein OData-Service kann optional Metadaten zur Verfügung stellen. Dies geschieht über ein „Service metadata document“. Dieses Dokument ist in erste Linie dazu da, um die automatische Generierung von Proxy-Klassen zu unterstützen, was die Entwicklung von OData-Clients erheblich vereinfacht.

Weiterhin kann es als Grundlage zur Visualisierung des Datenmodells eines OData-Service genutzt werden. Falls ein OData-Service ein Metadatendokument bereitstellt, erhält man es, indem man die „Query option“ `$metadata` an die „Service root“ URI anhängt. Das Metadaten-Dokument des oben benutzten OData-Beispiel-Service erhält man mit der URI `http://services.odata.org/OData/OData.svc/$metadata`. Spricht man diese URI über einen Browser an, so erhält man das in Abbildung 5 dargestellte Ergebnis.

Das Metadatenmodell wird als „Entity Data Model“ (EDM) bezeichnet und es hat starke Ähnlichkeit mit dem ER-Modell von Chen (s. o.). Das EDM ist Teil des ADO.NET Entity Frameworks, einem Werkzeug zur objektrelationalen Abbildung, das von Microsoft mit dem .NET Framework 3.5, Service Pack 1 im August 2008 eingeführt wurde. Die Beschreibung von Daten im Rahmen des EDM geschieht in einem XML-Dialekt, der als „Conceptual Schema Definition Language“ (CDSL) bezeichnet wird.

Das zentrale Konzept im EDM sind Entitäten und Assoziationen zwischen Entitäten. Entitäten sind Instanzen eines „entity type“, z. B. „Product“ in Abbildung 5. Sie bestehen aus einem Schlüssel („key“) und haben Eigenschaften („properties“), die aus einen Namen und Datentyp bestehen. Die Datentypen können primitiv („primitive type“) oder komplex („complex type“) sein. Komplexe Datentypen bestehen ebenfalls aus Eigenschaften, haben aber keinen Schlüssel. Sie können nur als Eigenschaft einer Entität existieren.

Die Eigenschaft „ReleaseDate“ der Entität „Product“ in Abbildung 5 ist z. B. der primitive Datentyp `Edm.DateTime`. Andere primitive Datentypen im EDM sind `Edm.Int32`, `Edm.Decimal`, `Edm.String`, …, also die üblichen primitiven Datentypen, wie man sie aus Programmiersprachen kennt. Weiter gibt es Navigationseigenschaften („navigation properties“), die an eine bestimmte Assoziation gebunden sind und über die man zu einer anderen Entität navigieren kann. Die Entität „Product“ in Abbildung 5 hat die Navigationseigenschaft „Supplier“, über die ein Produkt mit seinem Lieferanten verknüpft ist.

Eine genauere Beschreibung des EDM würde den Rahmen dieses Artikels sprengen. Eine detailliertere Darstellung ist in

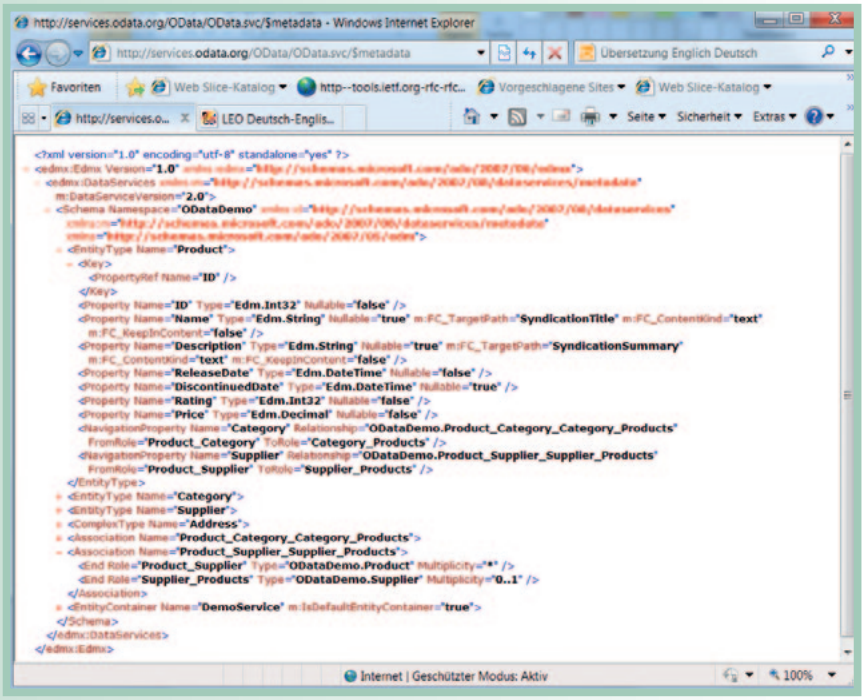

Abb. 5: Ergebnis der Browser-Abfrage **http://services.odata.org/OData/OData.svc/$metadata**. Die Darstellung der Entitäten „Category", „Supplier" und „Address" wurde nicht expandiert, um die Größe der Abbildung zu reduzieren

[ODataOver], Kapitel 4, zu finden. Eine gute Einführung in das EDM wird auch in [Ler10], Kapitel 2, Seite 19 - 47, gegeben. Eine intuitive Methode, das EDM zu erkunden, ist es, die Service- und Metadaten-Dokumente existierender OData-Services mit dem Web-Browser zu analysieren. Sehr hilfreich sind dabei auch die Werkzeuge OData Explorer und Open Data Protocol Visualizer (siehe unten).

Auf bereits bestehende OData-Services wird im Folgenden eingegangen.

Existierende OData-Anbieter und OData-Clients

Aktuell existieren schon einige Anbieter (OData Producer) von OData-Services. Im kommerziellen Bereich gibt es sowohl Softwareprodukte, die OData-Services implementiert haben, als auch Unternehmen, die OData-Services nutzen, um z. B. Katalogdaten verfügbar zu machen. Im öffentlichen Bereich gibt es die Open Governement Data Initiative (OGDI), welche Daten aus dem Bereich der öffentlichen Verwaltung über OData-Services bereitstellen bzw. bereitstellen wollen.

Im Bereich der Wissenschaft gibt es eine analoge Bestrebung, die Open Science Data Initiative (OSDI).

OData-Clients sind Applikationen, die über eingebaute oder nachrüstbare Schnittstellen zu OData-Services verfügen. Im Folgenden werden einige OData-Anbieter und -Clients vorgestellt, ohne dabei einen Anspruch auf Vollständigkeit zu erheben.

Applikationen als OData-Anbieter

In der folgenden Aufzählung sind einige Applikationen aufgelistet, die heute bereits als OData-Anbieter genutzt werden können:

- Der IBM WebSphereeXtremeScale REST data service ist ein zu OData konformer Service. Details sind unter **http://www.ibm.com/developerworks/websphere/downloads/xs_rest_service.html** zu finden.
- Die Cloud-Datenbank Microsoft SQL Azure sowie Microsoft Windows Azure Table Storage sind OData-Anbieter. Einzelheiten findet man unter **http://msdn.microsoft.com/de-de/windowsazure/default.asp**.
- Microsoft SharePoint 2010 und SQL Server 2008 R2 Reporting Services besitzen ebenfalls OData-Schnittstellen, mit denen sie Daten bzw. Datenbankreports verfügbar machen.

Kommerzielle und öffentliche OData-Anbieter

Beispiele für kommerzielle und öffentliche OData-Anbieter sind:

- Netflix ist ein Unternehmen, welches in den USA und Kanada Video-on-Demand über das Internet und den Verleih von Video DVDs und Blue-Ray Discs anbietet. Netflix hat seinen Katalog über einen OData-Service zugänglich gemacht. Die „Service-root" ist **http://odata.netflix.com/v1/Catalog/**.
- Die Stadt Vancouver in Kanada stellt Daten über öffentliche Einrichtungen als OData-Servicebereit.Die „Service-root"ist **http://vancouverdataservice.cloudapp.net/v1/**.

Applikationen als OData-Clients

Folgende Software ist aktuell in der Lage, OData-Services zu konsumieren:

- Jeder moderne Web-Browser, wie z. B. Microsoft IE 8, kann AtomPub-basierte Daten anzeigen.
- PowerPivot, ein Plug-in für Excel 2010 zur Analyse großer Datenmengen, kann OData-Services konsumieren, siehe dazu z. B. **http://powerpivot-info.com/post/341-powerpivot-odata-and-the-democratization-of-business-intelligence**.
- Der Open Data Protocol Visualizer ist ein Werkzeug für Microsoft Visual Studio 2010, mit dem das Datenmodell, welches ein OData-Service veröffentlicht, visualisiert werden kann. Das Diagramm in Abbildung 2 wurde damit erstellt. Das Tool ist unter **http://visualstudiogallery.msdn.microsoft.com/en-us/f4ac856a-796e-4d78-9a3d-0120d8137722** frei verfügbar.
- Der OData Explorer ist eine in Silverlight programmierte Browser-Applikation, mit der man das Datenmodell existierender OData-Services untersuchen kann. Er ist über die URL **http://www.silverlight.net/content/samples/odataexplorer/** aufrufbar.

Es gibt weiterhin eine Reihe von SDKs für verschiedene Programmiersprachen, mit denen eigene OData-Anbieter und Clients implementiert werden können. Aktualisierte Tabellen mit Links zu diesen SDKs, OData-Anbietern und OData-Clients sind in [OData] unter „Producers" respektive „Consumers" zu finden. Zwei solche SDKs für Java werden im nächsten Abschnitt vorgestellt.

OData Java SDKs

OData basiert auf den etablierten Technologien HTTP, XML, AtomPub und JSON, für die es schon im Java SDK selbst Unterstützung gibt oder über entsprechende Frameworks. Es ist also damit grundsätzlich möglich, OData-Clients oder OData-Anbieter zu implementieren. Einfacher ist es, wenn man durch entsprechende Frameworks unterstützt wird, welche einem einen großen Teil Programmierarbeit abnehmen, indem sie die

Entwicklung von OData-Clients durch Codegeneratoren unterstützen, welche die Metadaten-Dokumente nutzen und die Programmierung von OData-Anbietern durch entsprechende Klassenmodelle erleichtern. Aktuell (November 2010) existieren zwei solche Frameworks: *Restlet OData Extension* und *OData4J*. Beide sind frei verfügbar und quelloffen.

Restlet OData Extension

Dieses Framework ist eine Erweiterung des bekannten Restlet-Frameworks zur Vereinfachung der Implementierung von RESTful Services in Java. Die Restlet OData Extensions unterstützen in der aktuellen Version Restlet 2.0.2 nur die Implementierung von OData-Clients. Die Entwicklung von Software auf der Basis von Restlet wird umfassend in [LouBoi09] behandelt.

Zu den OData Extensions finden Sie alle wichtigen Informationen unter `http://www.restlet.org`. Unter `http://wiki.restlet.org/docs_2.0/13-restlet/28-restlet/287-restlet.html` finden Sie ein Tutorium. Die Version 2.0.2 kann unter `http://www.restlet.org/downloads/testing` heruntergeladen werden.

OData4J

OData4J ist ein weiteres Framework zur Implementierung sowohl von OData-Clients als auch von OData-Anbietern in Java. Es basiert auf *Jersey* (`https://jersey.dev.java.net/`) und nutzt *Joda Time* (`http://joda-time.sourceforge.net/`). OData4J unterstützt laut Dokumentation die Entwicklung von OData-Clients auf Android und die Implementierung von OData-Anbietern auf der Google AppEngine. OData4J ist aktuell in der Version 0.3 verfügbar. Es kann als zip-Archiv von der unten genannten Web-Seite heruntergeladen werden.

Entpackt man das zip-Archiv, so entsteht ein Verzeichnis mit dem Namen *odata4j-archive-x.y* (aktuell x.y = 0.3) mit den zwei Unterverzeichnissen *bundles* und *odata4j*. Alle benötigten jar-Dateien sind in den Unterverzeichnissen enthalten. Die Dokumentation befindet sich im Unterverzeichnis *javadoc* von *odata4j*. Das Verzeichnis *bundles* enthält die jar-Dateien *odata4j-bundle-0.3.jar* und *odata4j-clientbundle-0.3.jar*.

Will man nur OData-Clients entwickeln, so reicht es, *odata4j-clientbundle-0.3.jar* in den CLASSPATH aufzunehmen. Einfügen von *odata4j-bundle-0.3.jar* in den CLASSPATH ist hinreichend für die Implementierung von Clients und Anbietern. Damit man die Beispiele nutzen kann, benötigt man unbedingt die folgenden Java-Dateien: *BaseExample.java*, *ODataEndpoints.java*, *ProducerUtil.java* und *WhitelistFilter.java*. Die Dateien enthalten Hilfsklassen,welche unter anderem die Ausgabe von Entitäten und die Implementierung von OData-Anbietern vereinfachen.

Die Beispiele sind in dem oben genannten zip-Archiv nicht enthalten. Sie müssen extra von der unten angegebenen URL heruntergeladen werden. Bei den Beispielen sind unter anderem ein OData-Client (*NetflixConsumerExample.java*), der den oben genannten netflix-Service nutzt, sowie ein OData-Anbieter (*InMemoryProducerExample.java*), der lokal gestartet werden kann. Er stellt Systemdaten, wie die Anzahl der Java-Threads, die auf der JVM gerade ausgeführt werden, die Liste der Umgebungsvariablen in seinem Ausführungskontext, die Liste der Dateien in seinem Startverzeichnis usw., als OData-Service zur Verfügung. Der netflix OData-Client funktioniert einwandfrei.

Weiterhin wurde ein einfacher OData-Producer mit .NET 4.0 und Visual Studio 2010 programmiert und dafür ein Java-Client mit OData4J implementiert, der die Daten des .NET 4.0 Data-Producers ohne Probleme lesen und schreiben konnte.

Der OData-Anbieter *InMemoryProducerExample* wurde mit verschiedenen Microsoft OData-Clients (Open Data Protocol Visualizer, PowerPivot for Excel 2010, und individueller NET 4.0 OData-Client in C#) getestet, wobei der Server korrekt funktionierte. Die hier beschriebenen Beispiele und Tests wurden mit dem Java SE Development Kit Version 1.6.0.21 durchgeführt.

Insgesamt macht OData4J für eine Version 0.3 einen brauchbaren Eindruck. Dass man nur eine jar-Datei in den CLASSPATH aufnehmen muss, um mit OData4J zu arbeiten, vereinfacht das Ganze und schließt einige Fehlerquellen aus. Die Dokumentation lässt allerdings noch viele Wünsche offen. Man muss sich aktuell die Anwendung von OData4J anhand der Beispiele erarbeiten, was machbar ist, da diese sehr übersichtlich sind. Es lohnt sich also, das OData4J-Projekt im Auge zu behalten, wenn man in Zukunft anstrebt, OData-Anbieter mit Java zu realisieren.

Alle wichtigen Informationen zu diesem Java OData SDK finden Sie unter `http://code.google.com/p/odata4j/`, die Beispiele unter `http://code.google.com/p/odata4j/source/browse/tags/0.3/odata4j-core/test/org/odata4j/examples/`.

Zusammenfassung und Ausblick

OData ermöglicht es, „RESTful" Services zu entwerfen und Daten plattform- und programmiersprachenunabhängig im Inter- oder Intranet verfügbar zu machen. Es setzt auf den etablierten Standards bzw. Spezifikationen HTTP, XML, AtomPub und JSON auf und unterstützt damit die einfache Implementierung von leicht handhabbaren OData APIs auf allen modernen Applikationsentwicklungsplattformen, welche den Softwareentwicklern helfen, bestehende OData-Services zu nutzen oder neue zu implementieren.

OData hat also die Chance, das webbasierte Äquivalent zu ODBC und JDBC zu werden, vor allem wenn man bedenkt, dass durch die Verbreitung von mobilen Endgeräten auf der Basis von Windows Phone 7, iPhone, Android usw. der einfache und standardisierte Zugriff auf Daten immer wichtiger wird. OData als etablierte Methode für den Zugriff auf Daten im Inter- und Intranet sollte damit auch für die Entwickler von Apps für iPhone, Windows Phone 7 usw. große Vorteile bringen.

Literatur und Links

[Ler10] J. Lerman, Programming Entity Framework, 2. Auflage, O'Reilly 2010

[LouBoi09] J. Louvel, Th. Boileau, Restlet in Action, Manning Early Access Edition, 2009, `http://www.manning.com/louvel/`

[OData] Open Data Protocol, `http://www.odata.org/`

[ODataOver] Open Data Protocol Overview, `http://www.odata.org/developers/protocols/overview`

[ODataQA] Open Data Protocol: Fragen und Antworten, `http://msdn.microsoft.com/en-us/data/ee844254.aspx`

[Til09] St. Tilkov, REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien, dpunkt Verlag, 2009

[WikiERM] Entity-Relationship-Modell, `http://de.wikipedia.org/wiki/ER-Modell`

Klaus Rohe arbeitet bei der Microsoft Deutschland GmbH. E-Mail: klaus.rohe@microsoft.com