A UML Profile for OData Web APIs

Hamza Ed-douibi[1(✉)], Javier Luis Cánovas Izquierdo[1], and Jordi Cabot[1,2]

[1] UOC, Barcelona, Spain
{hed-douibi,jcanovasi}@uoc.edu
[2] ICREA, Barcelona, Spain
jordi.cabot@icrea.cat

Abstract. More and more individuals and organizations are making their data available online publicly, resulting in a growing market of technologies and services to help consume data and extract its real value. One of the several ways to publish data on the Web is via Web APIs. Unlike other approaches like RDF, Web APIs provide a simple way to query structured data by relying only on the HTTP protocol. Standards and frameworks such as Open API or API Blueprint offer a way to create Web APIs but OData stands out from the rest as it is specifically tailored to deal with data sources. However, creating an OData Web API is a hard and time-consuming task for data providers as they have to choose between relying on commercial solutions, which are heavy and require a deep knowledge of their corresponding platforms, or create a customized solution to share their data. We propose an approach that leverages on model-driven techniques to facilitate the development of OData Web APIs. The approach relies on a UML profile for OData allowing to annotate a UML class diagram with OData stereotypes. In this paper we describe the profile and show how class diagrams can be automatically annotated with such profile.

Keywords: UML · OData · Web API

1 Introduction

Recent years have seen an explosion of data available online via Web APIs, coming from both the public sector and private sources. Unlike other approaches like RDF, Web APIs provide a simple way to query structured data by relying only on the HTTP protocol. The increasing number of Web APIs has actually led to an explosion of specialized applications that combine data from different sources to provide insights on specific topics not visible at first glance, thus contributing to the growth of data economy.

While standards and frameworks such as Open API[1] or API Blueprint[2] offer a way to create Web APIs, the Open Data Protocol (OData)[3] is specifically

[1] https://www.openapis.org/.
[2] https://apiblueprint.org/.
[3] http://www.odata.org/.

J. Cabot et al. (Eds.): ICWE 2017, LNCS 10360, pp. 420–428, 2017.
DOI: 10.1007/978-3-319-60131-1_28

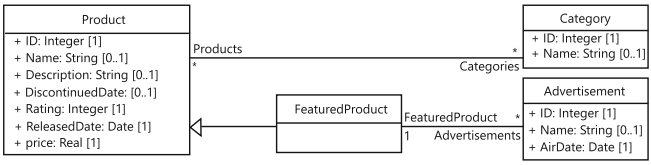

Fig. 1. UML class diagram of the running example.

tailored to deal with data sources. Thus, in the last years, OData protocol has been accepted as the favored standard to publish datasets as Web APIs. As a result, many commercial infrastructures have integrated OData to their products (e.g., SAP, IBM WebSphere, JBoss Data Virtualization).

OData enables the creation of data-centric Web APIs, which allow resources, identified using Uniform Resource Locators (URIs) and defined in a data model, to be published and edited by Web clients using simple HTTP messages. It defines also a small URL-based query language to identify and query the data described in the data model. The current version of OData (version 4.0) has been approved as OASIS standard [4]. However, creating an OData Web API is still a hard and time-consuming task for data providers as they have to choose between relying on commercial solutions, which are heavy and require a deep knowledge of their corresponding platforms, or create a customized solution to share their data.

Model-Driven Engineering (MDE) is a paradigm which emphasizes the use of models to raise the level of abstraction and automation in software development [10]. MDE aims to address platform complexity by using models and model transformations for the specification/generation of software artifacts. Thus, MDE techniques have been increasingly used to automate the generation of Web applications [2,3,6–9,11,12]. While these existing MDE approaches cover a variety of technologies (e.g. web services and ubiquitous applications), they lack of specific support for OData (and REST APIs in general, with very few exceptions [2,6,9]).

In this sense, our goal is to advance towards the definition of an MDE infrastructure for the generation (and reverse engineering) of OData applications. As a first step towards this vision, this paper presents a UML profile for OData that enables an easy definition of OData sources at the model level.

The remainder of this paper is structured as follows. Section 2 shows the running example used along the paper. Section 3 presents the OData profile and Sect. 4 presents the rules to generate default profile definitions. Finally, Sect. 5 concludes the paper and presents some future work.

2 Running Example

We define a simple OData Web API of an online store as running example. This example is inspired in the official reference example of the OData community[4]. Figure 1 shows an excerpt of the UML class diagram for the Web API data model,

[4] http://services.odata.org/V4/OData/OData.svc/$metadata.

Listing 1. A simple OData Metadata Documents for the products service

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version
    ="4.0">
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="com
        .example.ODataDemo" Alias="ODataDemo">
      <EntityType Name="Product">
        <Key><PropertyRef Name="ID"/></Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false"/>...
        <NavigationProperty Name="Categories" Type="Collection(ODataDemo.
            Cotegory)" Partner="Products"/>
      </EntityType>
      <EntityType Name="Category">
        <Key><PropertyRef Name="ID"/></Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
        <Property Name="Name" Type="Edm.String"/>
        <NavigationProperty Name="Products" Type="Collection(ODataDemo.
            Product)" Partner="Categories"/>
      </EntityType>
      <EntityType Name="FeaturedProduct" baseType="Product">...</
          EntityType>
      <EntityType Name="Advertisement">...</EntityType>
      <EntityContainer Name="ODataDemoService">
        <EntitySet Name="Products" EntityType="ODataDemo.Product">
          <NavigationPropertyBinding Path="Categories" Target="Categories
              "/>
        </EntitySet>
       ...
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

which includes the classes: `Product` to represent products, `Category` to classify products, `FeaturedProduct` for premium products to be featured in commercials, and `Advertisement` which records the data about those commercials.

In OData data models are not expressed in UML but as XML metadata documents describing an Entity Data Model (EDM) using the Conceptual Schema Definition Language (CSDL) [5]. Web clients use this document to understand how to query and interact with the API using standard HTTP methods. Listing 1 shows an excerpt of the metadata document for the data model shown in Fig. 1. The `Schema` element describes the entity model exposed by the OData Web APIs and includes the entity types `Product` and `Category`, `FeaturedProduct`, and `Advertisement`, which also includes properties and navigation properties to describe primitive attributes and associations, respectively. The `Schema` element includes also an `EntityContainer` element defining the entity sets exposed by the service and therefore the entities that can be queried. In the following section, we will describe the OData profile which enables the generation of such file.

3 A UML Profile for OData

To formalize domain-specific knowledge, we can either create a new metamodel or extend an existing modeling language. Given the similarities between OData

and available concept in UML (specifically, UML class diagrams), we opted to use the UML extension mechanisms (providing stereotypes, tagged values, and constraints to adapt the UML metamodel to different platforms or domains) as the basis for our OData modeling language. Therefore, this section presents our OData profile for UML.

We organize the OData profile into two parts, namely: (i) the Entity Data Model (EDM) which describes the data exposed by an OData Web API, and (ii) the advanced configuration model, which defines additional characteristics or capabilities of OData Web APIs (i.e. what parts of the EDM can be modified, what permissions are needed,...).

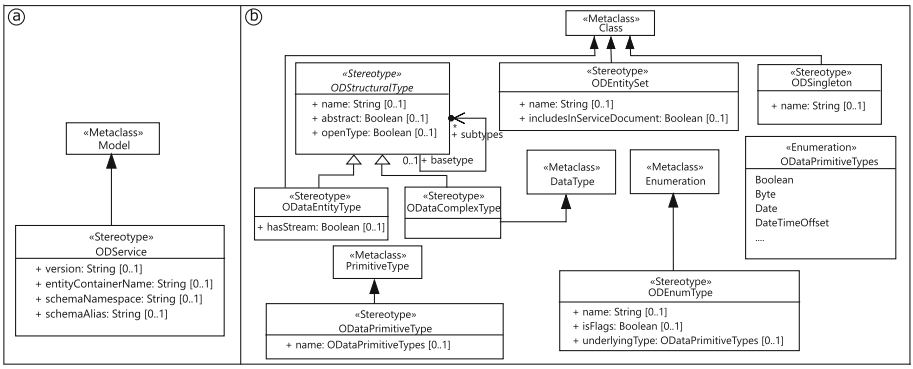

Fig. 2. OData profile: (a) the service wrapper and (b) data types elements.

3.1 The Entity Data Model

OData Service Wrapper. An OData Web API exposes a single entity model which may be distributed over several schemas, and should include an entity container that defines the resources exposed by the Web API. Figure 2a shows the extension of UML to define these elements. We consider that the entity model is defined in one schema, represented by the element *Model* of a UML class diagram. Thus, `ODService` stereotype extends the metaclass `Model` and includes: the version the OData specification, the namespace of the schema (e.g., `com.example.ODataDemo`), an alias for the schema namespace (e.g., `ODataDemo`), and the name of the entity container (e.g., `ODataService`).

Data Types. An OData entity model defines data types in terms of structural types, enumerations, and primitive types. There are two kinds of structural types: *entity types* and *complex types*. An *entity type* is a named structured type which defines the properties and relationships of an entity. *Entity types* are mapped to the concept *Class* in a UML model. A *complex type* is a named structural type consisting of a set of properties. *Complex types* are mapped to the concept *Data type* in a UML model.

Figure 2b shows the stereotypes related to data types and their mapping with UML concepts. The abstract stereotype `ODStructuralType` defines the common features of all the structural types and includes a name, a property indicating whether the structural type cannot be instantiated (i.e., `abstract` property), and a property indicating whether undeclared properties are allowed (i.e., `openType` property[5]).

`ODStructuralType` supports also the concept of inheritance by allowing the declaration of a base structural type (i.e., `basetype` association). The stereotypes `ODEntityType` and `ODComplexType` inherit from `ODStructureType` and extend the metaclasses `Class` and `DataType`, respectively. Additionally `ODEntityType` includes the `hasStream` property, indicating if the entity is a media entity (e.g., photograph). The stereotype `ODPrimitiveType` extends the metaclass `PrimitiveType` and includes a name which corresponds to the associated OData primitive type (e.g., Binary, Boolean, etc.). The stereoType `ODEnumType` extends the metaclass `Enumeration` and includes a name, a boolean property indicating whether more than one member may be selected at a time (i.e., `IsFlags` property), and the corresponding OData type.

The profile also allows modeling the entity sets and singletons exposed by the OData service. While an entity set can expose instances of the specified entity type, a singleton allows addressing a single entity directly from the entity container. These two concepts are materialized with the stereotypes `ODEnititySet` and `ODSingleton` which extend the metacalass `Class`.

Properties and Associations. Properties define the structure and the relationships in OData. Structural properties define the attributes of an entity type or a complex type where as navigation properties define associations between entity types. In UML the element *Property* is a *StructuralFeature* which, when related by *ownedAttribute* to a *Classifier* (other than *Association*), represents an attribute, and when related by *memberEnd* of an *Association*, represents an association end. Both structural properties and navigation properties are mapped to the concept *Property* in a UML model.

Figure 3 shows the stereotypes defining properties and associations. The stereotypes `ODProperty` and `ODnavigationProperty` represent a structural property and a navigation property, respectively. They both extend the metaclass `Property`. The stereotype `ODProperty` includes a name and several constraints to provide additional constraints about the value of the structural property (e.g., `nullable`, `maxLength` properties). Additionally, the stereotype `ODKey` inherits from `ODProperty` and defines a property as the key of the entity type (required for a an OData entity type). The stereotype `ODNavigationProperty` includes a name, a containment property, and a nullable property. The stereotype `ODNavigationPropertyBinding` extends also the metaclass `Property` and defines a navigation binding for the corresponding entity set.

[5] Open types entities allows clients to persist additional undeclared properties.

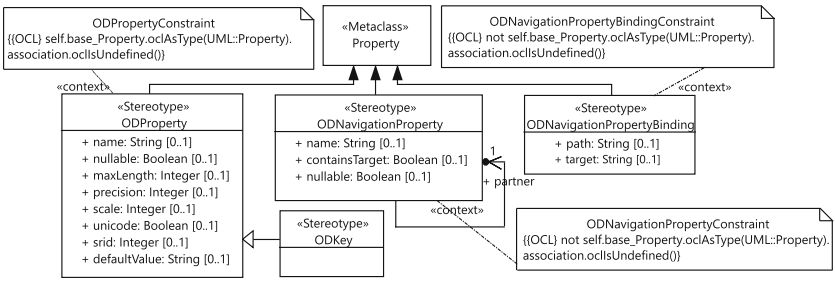

Fig. 3. Properties and associations stereotypes.

To ensure the validity of the applied stereotypes, we have enriched the profile with a set of constraints written using the Object Language (OCL) [1]. For instance, since the stereotypes related to properties and navigations properties extend all the metaclass `Property`, `ODPropertyConstraint` ensures that the stereotype `ODProperty` is applied to a UML property element representing an attribute.

3.2 Advanced Configuration of OData Web APIs

OData defines annotations to specify additional characteristics or capabilities of a metadata element (e.g., entity type, property) or information associated with a particular result (e.g., entity or property). For example, an annotation could be used to define whether a property is read-only. Annotations consist of a term (i.e., the namespace-qualified name of the annotation), a target (the element to which the term is applied), and a value. A set of related terms in a common namespace comprises a vocabulary. Our profile supports the three standardized vocabularies defined by OData, namely: the core vocabulary, capacity, and measures.

Figure 4 shows an excerpt of the profile defined for representing annotations. The stereotype `ODAnnotations` extends the metaclasses `Model`, `Class`, and `Property`, and has an association with `ODVocabulary`, thus allowing adding annotations according to the vocabularies. `ODVocabulary` is the root class of

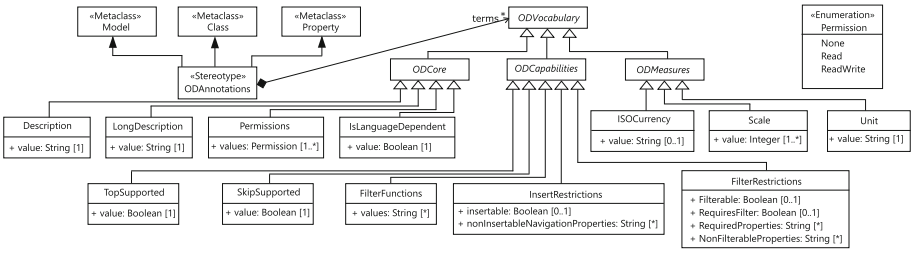

Fig. 4. Annotation and vocabulary stereotypes.

the hierarchy of vocabularies supported by the OData profile (i.e., core, capabilities, and measures vocabularies). OData profile defines (i) the `ODCore` hierarchy which includes the core vocabularies such as documentation (e.g., the class `Description`), permissions (i.e., the class `Permissions`, and localization (i.e., the data type `IsLanguageDependent`); (ii) the `ODCapabilities` hierarchy which is used to explicitly specify Web API capabilities (e.g., `TopSupported` for query capabilities or `InsertRestriction` for data modification); and (iii) the `ODMeasures` hierarchy to describe monetary amounts and measured quantities (e.g., `ISOCurency`).

4 Default Profile Generation

Our OData profile can be used to annotate any new or preexisting UML class diagram. Nevertheless, to simplify the application of our profile, we have also developed a model-to-model transformation that given an standard UML model, returns an annotated one by relying on a set of default heuristics that embed our knowledge on typical uml-to-odata design decisions. This annotated model can be regarded as just an initial option to bootstrap the process that the designer can then modify at will.

Table 1 summarizes our mapping strategies. From left to right, the columns of the table show (1) the involved UML element; (2) the conditions to apply an stereotype (if any), (3) the stereotype to be applied; and (4) the values of the stereotype properties. In a nutshell, each class is mapped to an entity type and is exposed as entity set, each attribute is mapped to a property, and each navigable association is mapped to a navigation property. Figure 5 shows the running example including some of the generated OData profile annotations. This first version of the class diagram can later be customized and used in other model-driven processes to fast prototyping OData Web APIs.

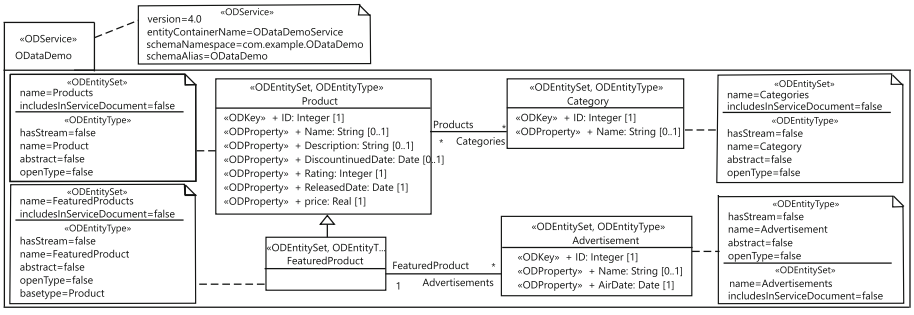

Fig. 5. UML class diagram of the running example annotated by the generator.

Table 1. Rules of the OData profile annotation generator.

UML element	Condition	Stereotype	Value
m: Model	-	ODService s	- s.version = "4.0" - s.entityContainerName= m.name+"Service" - s.schemaNamespace = "com.example."+m.name - s.schemaAlias = m.name
c: Class	-	ODEntityType et	-et.name = c.name -if c.abstract == true then et.abstract = true -if c.generalization contains t then et.basetype=ot (ot is the entity type of t)
		ODEntitySet es	- es.name = the plural form of c.name
p: Property	p is an class attribute OR a data type attribute	ODProperty op	- op.name = p.name - if p.lower == 1 then op.nullable = false -op.defaultValue = p.default
	p is an class attribute marked as key	ODKey ok	- ok.name = p.name
	p is an navigable association end	ODNavigationProperty np	- np.name = p.name - if p.lower == 1 then np.nullable = false - if p.aggregation ==composite then np.containsTarget = true
		ODNavigationPropertyBinding npb	- npb.path = p.name - npb.target = the name of the corresponding entity set of the association end
dt: DataType	-	ODComplexType ct	- ct.name = dt.name - if dt.abstract == true then ct.abstract = true - if dt.generalization contains t then ct.base=ot (ot is the complex type of t)
pt: PrimitiveType	-	ODPrimtiveType opt	- opt.value = the corresponding primitive type of pt.name
e: Enumeration	-	ODEnumType oe	- oe.name = e.name

5 Conclusion

In this paper we have presented a UML profile to model OData Web APIs and the corresponding annotation generator for any UML class diagram. We believe our approach is the first step to boost the model-based development of OData Web APIs, offering developers the opportunity to leverage on the plethora of modeling tools to define forward and reverse engineering to generate, visualize and manipulate OData sources. The OData profile along with the default profile generator are available as an Open Source Eclipse plugin[6]. The plugin repository includes also the steps to reproduce the running example.

As future work we aim at extending our profile in order to capture additional OData behavioral concepts such as functions and actions. We would also

[6] https://github.com/SOM-Research/OData.

like to integrate this profile with other web-based modeling languages like IFML (e.g. by enabling the use of OData-like data sources as part of the interface modeling components) in order to create a rich modeling environment combining front-end and back-end development. Finally, we plan to complement the profile support with model-to-text and model-to-model transformations to offer, for instance, the (semi)automatic code-generation of OData services from the annotated models.

Acknowledgment. This work has been supported by the Spanish government (TIN2016-75944-R project).

References

1. Cabot, J., Gogolla, M.: Object constraint language (OCL): a definitive guide. In: Formal Methods for Model-Driven Engineering, pp. 58–90 (2012)
2. Ed-Douibi, H., Izquierdo, J.L.C., Gómez, A., Tisi, M., Cabot, J.: EMF-REST: generation of restful APIs from models. In: SAC Symposium, pp. 1446–1453 (2016)
3. Fraternali, P.: Tools and approaches for developing data-intensive web applications: a survey. CSUR **31**(3), 227–263 (1999)
4. Pizzo, M., Handl, R., Zurmuehl, M.: Odata version 4.0 part 1: protocol. Technical report, OASIS (2014)
5. Pizzo, M., Handl, R., Zurmuehl, M.: Odata version 4.0 part 3: Common Schema Definition Language (CSDL). Technical report, OASIS (2014)
6. Rivero, J.M., Heil, S., Grigera, J., Gaedke, M., Rossi, G.: MockAPI: an agile approach supporting API-first web application development. In: ICWE Conference, pp. 7–21 (2013)
7. Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.): Web Engineering: Modelling and Implementing Web Applications. Human-Computer Interaction Series. Springer, London (2008)
8. Schwinger, W., Retschitzegger, W., Schauerhuber, A., Kappel, G., Wimmer, M., Pröll, B., Cachero Castro, C., Casteleyn, S., De Troyer, O., Fraternali, P., et al.: A survey on web modeling approaches for ubiquitous web applications. IJWIS **4**(3), 234–305 (2008)
9. Segura, Á.M., Cuadrado, J.S., de Lara, J.: ODaaS: towards the model-driven engineering of open data applications as data services. In: EDOCW Workshop, pp. 335–339 (2014)
10. Selic, B.: The pragmatics of model-driven development. IEEE Softw. **20**(5), 19–25 (2003)
11. Valderas, P., Pelechano, V.: A survey of requirements specification in model-driven development of web applications. TWEB **5**(2), 10 (2011)
12. Vallecillo, A., Koch, N., Cachero, C., Comai, S., Fraternali, P., Garrigós, I., Gómez, J., Kappel, G., Knapp, A., Matera, M., Meliá, S., Moreno, N., Pröll, B., Reiter, T., Retschitzegger, W., Rivera, J.E., Schauerhuber, A., Schwinger, W., Wimmer, M., Zhang, G.: MDWEnet: a practical approach to achieving interoperability of model-driven web engineering methods. In: MDWE Workshop, @ICWE (2007)