ODaaS: Towards the model-driven engineering of open data applications as data services

Ángel Mora Segura*, Jesús Sánchez Cuadrado*, Juan de Lara*
* Modelling and Software Engineering Group (http://www.miso.es)
Department of Computer Science
Universidad Autónoma de Madrid (Spain)
{Angel.MoraS, jesus.sanchez.cuadrado, Juan.deLara}@uam.es

***Abstract*—The *Data-as-a-Service* (DaaS, or Data Services) paradigm enables an on-demand, service-based access to data, relying on similar principles to Software-as-a-Service (SaaS). DaaS permits centralized data quality management, a uniform view and access to heterogeneous data, and enables exposing a richer, domain-specific data model to users.**

Within this context, we are witnessing a trend in institutions to make information public as *open data*. However, such information is normally released "as-is", in heterogeneous formats, requiring costly, ad-hoc pre-processing steps for cleansing and analysis of its underlying structure.

This paper proposes an adaptation of the DaaS paradigm for the construction of open data applications. For this purpose, we introduce an architecture based on Model-Driven Engineering (MDE), consisting of (i) multi-level modelling for the description of domains, based on generic meta-models, (ii) a library of injectors to bring data on demand from heterogeneous sources into the MDE technical space, and (iii) a REST-based infrastructure to access the data services. This work presents the architecture of such framework and the first steps in its realization.

Keywords*—*Model-driven engineering, Multi-level modelling, Data services, Open data, Open Data as a Service (ODaaS)

I. Introduction

The Data-as-a-Service (DaaS) approach is a variant of the Service-Oriented Computing (SOC) paradigm that allows suppliers to expose their internal data using a richer data model than the one in which the data is originally stored [4]. At the same time, there is a trend among governments, institutions and even enterprises to make all sort of information public, like budgets, expenses or economic data. Typically, such data is released as *open data* that can be freely used and distributed [12]. However, it is not enough to distribute the data, but it has to be made available in a form that makes it useful to consumers. Hence, providing services for data is an appealing mechanism to increase its value.

The engineering of data services for open, heterogenous data poses a series of challenges due to the large variety of data formats, the lack of an underlying data model in many cases, the heterogenity of the data and the data sources, their dynamic nature in some cases, and a bigger probability of unavailability, among other factors. This requires a different approach from existing proposals, which normally focus on business scenarios where data is kept in some internal, controlled storage.

Our proposal to tackle this problem is based on Model-Driven Engineering (MDE) [19], a development paradigm exploiting the use of domain models to raise the level of abstraction and automation at which software is developed. MDE is based on two key elements: meta-modelling and model transformations. Meta-modelling defines the structure and semantics of domain models, while transformations permit the automatic model manipulation for different purposes, like refactoring, model querying, code generation, language mappings, and conversions between technological spaces [3].

In this position paper we describe the architecture of our proposal, which is based on three main elements. Firstly, we use multi-level modelling [2] to characterize the data in different domains. We show how this approach to meta-modelling allows a hierarchical classification of the different data domains, relying on a top-level layer of reusable, generic meta-models. We also discuss the challenges arising when applying multi-level modelling in practice in this context. Secondly, to cope with the inherent heterogenity regarding data sources and formats we define the notion of domain injector, which allows mapping data in some format into a semantically rich model. Thus, we are defining a family of domain injectors to deal with some common formats and data sources. Finally, a data service is accessible through a generic REST API which exposes the data model described by the domain models. In this paper, we motivate this approach, describe the architecture, and provide an overview of our ongoing implementation.

Paper Organization. Section II presents an overview of the architecture. Its main elements are described in the subsequent three sections: Section III introduces our multi-level modelling organization, Section IV describes our approach to build domain injectors. Section V describes our prototype implementation and discusses some applications of the framework. Section VI compares our approach with related work and Section VII presents the conclusions and lines for future work.

II. ODaaS: A DaaS Architecture for Open-Data Applications

A. Motivation and Requirements

The *open data movement* advocates the availability of certain data, typically public data, coming from a range of sources such as scientific institutions or governments. Notably, the possibility for citizens to oversee data related to government behavior (national, regional or municipal) or to the performance of public services will lead to the so-called open-government. Examples of open data provided by governments

include transport information, criminality reports, accountability and expenses [9].

There is a growing interest in applications making use of open data, as can be witnessed by the European Union initiative[1], the emergence of events like the *Open Data Challenge*[2], the *Open Data Days*[3], and others sponsored by local councils like the *Ottawa Open Data App Contest*[4]. Thus, data providers require crowdsourcing ideas leading to useful applications profiting from the large amount of delivered open data.

Nowadays, such applications need to be handcrafted by skilled people with highly specialized technical background. Our framework aims at facilitate the use of open data to both developers and users, driven by the following requirements.

1) **Dealing with heterogeneous data formats**. Open data are typically published in the most convenient way for the publisher. This leads to heterogeneous data formats which require a careful pre-processing before their use in applications. Mechanisms to facilitate data processing and giving data a homogenous structure are needed for data consumers to make the most of open data possibilities.
2) **Semantic data domains**. Open data applications need to interpret and combine heterogeneous data from different sources. Often such sources do not provide any semantic information, as the data is released as e.g., PDF or Excel documents. Semantically rich data domains must be defined in formalisms that facilitate mapping existing data to them, as well as creating new data domains, possibly derived from the existing ones.
3) **User interaction**. Applications with rich data analysis and visualization means (e.g., geolocalization of elements in interactive maps) need to be created for people to be able to make use of open data. This task should be as automated as possible, since new types of data are released continously. In addition, users should be given the possibility of interacting and contributing new relevant data (e.g., use of mobile devices to decorate existing data with opinions or ratings, or to report local problems in cities like potholes in roads, broken lights or graphities). For this latter issue, it would be desirable to incorporate connections with social networks, like Twitter, given the wide use of such networks nowadays [15].

B. Architecture

In order to cope with the previous requirements, we propose the architecture shown in Fig. 1. We define our approach Open-Data-as-a-Service (ODaaS), as it delivers open-data in the form of services. The rationale is to make uniform different data sources, by their representation in the form of models, within the MDE technical space. For example, open data is typically made available in heterogeneous documents (Excel, XML, CSV, PDF), while many companies offer open APIs to perform predefined queries. Other data may be located in relational data bases, or be only available as streaming data (Tweets, or sensor information).

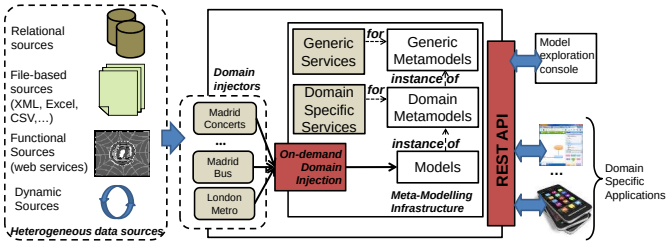

Fig. 1: Architecture of our approach.

In our proposal, we internally represent data as models, conformant to a well defined, semantically rich domain meta-model. Hence, as part of our methodology, we use meta-models to represent different domains, like city transportation or cultural events. Please note that representing the data in the form of models goes beyond a simple change of format, but it implies giving semantics to the data through these domain meta-models. As we will see in Section III, our domain meta-models are built by instantiating predefined generic meta-models, useful to represent, e.g., temporal information, geopositions or personal information. The advantage is that we can define generic operations over these meta-models (e.g., like representing geopositions in maps, reasoning about temporal information, etc.), so as to make them available to instances of any domain meta-model. Additionally, more specific services can be defined over the domain meta-models.

To import data into the MDE technical space, we use *domain injectors*. These take data in a certain format (XML, JSON), from some platform (file system, web service, database) and produce a model conformant to some domain meta-model (e.g., transportation). Hence, injectors are in charge of giving semantics to externally acquired data. As we will see in Section IV, domain injectors are made of two parts. The first one is a data injector that reads data in a certain format from some platform, and transform it into a "neutral" model. The second is a mapping (a model-to-model transformation [5]), translating such a neutral model into a model conformant to some domain meta-model. To facilitate the construction of domain injectors, reusable libraries of data injectors are needed, which we have started to build.

Another key point is that data can be dynamically fetched from data sources on demand. Hence, if the system receives a request for some model elements that are not internally present yet, a data retrieval component activates the corresponding domain injector to retrieve the data from the external source.

We make the system accessible as a REST service. We foresee both generic applications, like a console to discover and use the different data services, and domain-specific applications (e.g., a mobile application for a traffic information system aggregating multiple sources). By relying on MDE, appropriate meta-models and code generators can be built to synthesize the final application.

Altogether, MDE is appropriate in this context because it allows characterizing the data in a specific domain by means of meta-models, its validation and cleansing by checking its

[1] http://open-data.europa.eu/
[2] http://opendatachallenge.org/
[3] http://opendataday.org/
[4] http://www.apps4ottawa.ca/

conformance to integrity constraints, and using transformation technology for data mappings. Thus, data sets from different sources, but within the same domain, will be mapped to the same meta-model, making them compatible and homogeneous. Moreover, different services, like visualization, interaction and reasoning services can be defined generically, based on the concepts the domain meta-models are built with (e.g., temporal, geographical, personal). By selecting and combining different domain injectors, domains and services, applications using such data services can be generated using MDE.

Next, we detail the different elements of this architecture.

III. Domain modelling

In order to obtain a uniform view of heterogeneous data, we convert such data into a model, conformant to a meta-model. Meta-models make explicit the structure of the data, its static semantics (integrity constraints), and allow their automated manipulation and reasoning. In the following we describe our meta-modelling architecture, and the challenges imposed by its use in a DaaS based approach to address open data challenges.

A. Multi-level modelling

We propose the use of multi-level modelling [2] to allow a stratified classification of concepts and their successive refinement. Differently from standard meta-modelling approaches, like those based on the Meta-Object Facility (MOF) [11], multi-level modelling allows multiple classification levels (i.e., modelling at an arbitrary number of meta-levels). The left of Fig. 2 shows the multi-level organization we propose for ODaaS. At the top meta-level, generic meta-models describe general concepts, like geopositions, temporal relations or personal information. Such meta-models are used to describe particular domain meta-models, which in turn are instantiated with particular domain data. This organization is not limited to three meta-levels, as some domain meta-models may require refinemenent at subsequent meta-levels.

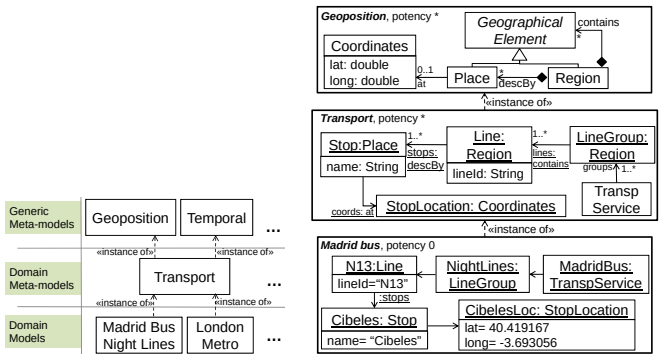

Fig. 2: Multi-level organization (left). Multi-level modelling for the transportation domain (right).

The right of Fig. 2 shows an excerpt of a multi-level model for the transportation domain. At the top, there is a simplified model (*Geoposition*) with concepts like *Place* or *Coordinates* to represent elements in the space. Such generic model can be instantiated to build a description of a transportation system. Some of the elements of this model are instances of the model above, like *Stop*, which is an instance of *Place*. However, in multi-level modelling [6] we allow *linguistic extensions*, elements that do not have a type in the meta-model above. Technically this is possible by the use of a dual linguistic/ontological typing [2]. All elements have linguistic typing, while some of them (like *TranspService*) may lack an ontological type. Similarly, attribute types with no ontological type are allowed, like *name* and *lineId* in *Stop* and *Line*. Finally, we can instantiate this model, to describe the particular configuration of the Madrid bus lines, at the bottom of the figure.

Some elements in the middle model (like *Line*) are instances of elements at the level above (*Line* is an instance of *Region*), and types with respect to the model at the level below (*Line* is a type for *N13*). Hence, as in multi-level modelling, elements may have both a type (class) and an instance (object) facet at the same time, they are called *clabjects* [2].

The advantage of this approach is that operations defined over the top-most meta-models become applicable any number of meta-levels below. For example, operations over the *Geoposition* meta-model may include representing a model in a map (by means of code generation), finding the *Place*s close to a given one, or converting between different coordinates systems (for simplicity the figure assumes just the latitude/longitude system). Hence, by using *Geoposition* to describe the transport domain meta-model, operations defined over *Geoposition* become applicable to instances of *Transport*. In practice, the transport meta-model is also built with a generic meta-model for describing temporal information, enabling e.g., the representation and reasoning about time points and intervals, but we do not show these details for the sake of simplicity.

Next, we analyse the challenges of using multi-level modelling for ODaaS.

B. Challenges of multi-level modelling for ODaaS

In multi-level modelling, clabjects are decorated with a *potency* [2], indicating the number of meta-levels they can be instantiated for. It is a natural number that decreases at each meta-level the clabject is instantiated. In this way, if a clabject has potency 2, its instances will have potency 1, and instances of these will have potency 0. A clabject with potency 0 cannot be instantiated further. Potency also applies to attributes, and is a way to characterize the structure of indirect instances of an element, several elements below. In the example, we may want to indicate that instances of *Coordinates* two levels below (like *CibelesLoc*) should have slots *lat* and *long*, and so we would need to assign such attributes potency 2. If an element does not receive a potency, it takes the one of its container (attributes take the potency of their owner clabjects if they do not declare one), and ultimately from their container models. Standard meta-modelling frameworks like MOF can be seen as a particular case of this approach, where meta-model elements have potency 1, and model elements potency 0.

Multi-level modelling based on potency needs to anticipate an arbitrary, but fixed number of meta-levels for each modelling problem, to assign a concrete potency to the elements in the top-level meta-model. However, this would be a limitation in the ODaaS domain, because at the top-level we have generic models (for *Geoposition*, *Temporal* and *Personal* information, etc.), which could be instantiated a different number of times

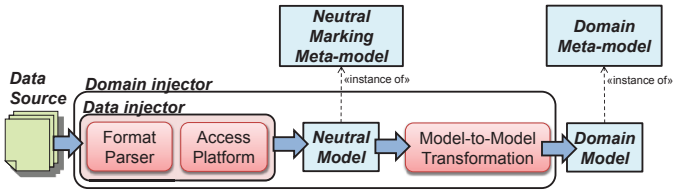

Fig. 3: Scheme of a domain injector.

depending on the domain. For example, we might have *Cultural events* as domain meta-model, to be refined into *Classical Music Festivals* one level below, and then a bottom level with information of the different festivals.

Hence, as also proposed by [1] in a different context, we allow the potency to be *unbounded*, written "*". Instantiating a clabject with unbounded potency may lead to an element with unbounded potency, or with a potency equal to a natural number. This is the approach followed in Fig. 2, where both the *Geoposition* and the *Transport* meta-models have unbounded potency. This enables refining *Transport*, should one wish to make explicit specific structures for some transportation system (e.g., a domain meta-model of a multi-mode transportation system may refine the *Transport* meta-model with the transportation type, prices, crossings, line operators and so on).

In addition, it may be needed to allow clabjects to have an arbitrary number of (ontological) types, and not just one. This would allow a *Stop* to be an instance of *Place* but also of *TemporalInterval*, to account for the bus timetable. In standard approaches, like MOF, this could be emulated by creating an intermediate class inheriting from the different classes to be instantiated (*Place* and *TemporalInterval*), and instantiating such intermediate class. However, this workaround does not scale, as intermediate classes would be needed for all possible combination of classes in the meta-models. Interestingly, the OMG has proposed an extension of the MOF with such feature in the so called MOF support for Semantic Structures (SMOF) [10], driven by needs in Ontology languages. However, still SMOF falls short in this line, as it needs to statically indicate the classes that can be combined together.

IV. Domain Injectors

Our architecture needs to transform external data into the MDE technical space, as an instance of a particular domain meta-model. For this purpose we propose the construction of *domain injectors*, whose structure is shown in Fig. 3.

A *domain injector* is made of two parts. First a *data injector* that brings data from a particular platform (a web service, a file) and format (XML, JSON). The data injector already creates a model, conformant to a domain-neutral meta-model (a *marking* meta-model, enabling a uniform representation of data in different formats). Data injectors are generic, and independent of the domain. In this way, we make available a library of reusable parsers for different formats (CSV, XML, JSON, Excel, etc) and platforms.

Secondly, a domain injector also includes a mapping into a particular domain (e.g., *Transportation*), which is currently realized as a model-to-model transformation from the marking meta-model into the domain meta-model. While data injectors are generic, this mapping is dependent on the particular domain meta-model. In our current implementation the model-to-model transformation needs to be written by hand, but our goal is to generate such transformation in a user-friendly way, using "by example" techniques [18].

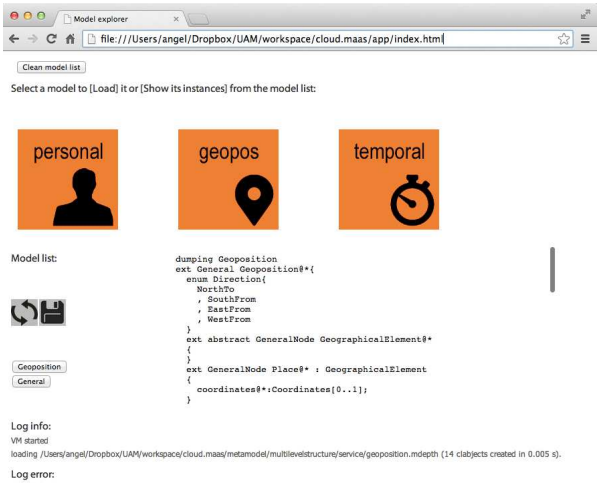

Fig. 4: The data service console.

V. Prototype Implementation

We have built a prototype implementation of the described architecture, based on MetaDepth [6] as a supporting tool for multi-level modelling. MetaDepth is integrated with the Epsilon model management languages[5] so that model-to-model transformations, in-place transformations, queries and code generators can be defined. Data services are exposed using a REST API, and we use Jersey[6] for that purpose. We have implemented some domain injectors for the *Transportation*, *Concerts* and *Historic Places* domains.

On top of such prototype, we have built a generic web application, which can be used as a console to explore the different data services, invoking its querys and browsing the resulting models. A screenshot is shown in Fig. 4. The figure shows an excerpt of the *Geoposition* meta-model, in MetaDepth format.

VI. Related Work

The interest in profiting from data of a different nature (business data, open data, social data) can be witnessed from the emergence of marketplaces of different kinds of data [14]. However, describing data services is still a challenging issue, since SOC related standards, such as WSDL [21], are not rich enough to describe the underlying data model of a service [4].

The Open Data Procotol [17] (OData) is a data access protocol which uses an object-oriented entity-relationship model, and defines a mapping between CRUD operations and a REST service to access the data model. It also provides means to query the service metadata (i.e., the data model structure). We

[5] http://www.eclipse.org/epsilon/
[6] https://jersey.java.net/

are studying a possible alignment of our query service with OData's REST service.

In [20] a proposal, called DEMODS, to describe a DaaS is presented. In DEMODS the data service API is decoupled from the data asset description. In our case, we propose a mapping from the data model to the API. DEMODS relies on external information models to describe the data assets (e.g., OData, OpenAnnotation), but it does not provide any special support for them, beyond links to the description documents. We use multi-level modeling as a pervasive element of our approach. However, exportation facilities to other formalisms are possible. The authors identify the challenge of "a common agreement in the definition of data domains and data categories", and we have shown how multi-level modeling allows us to naturally address it.

Ontologies could be used as an alternative to multi-level modelling. However, by remaining in the MDE technical space, we can use languages and tools for model manipulation, like transformations, code generation, and in-place transformations. In any case, we are currently working on an integration of METADEPTH and ontologies.

With respect to using a DaaS approach to expose open data, some of the challenges involved are commented in [7]. In this line, some efforts for the systematic use of open data are starting to emerge. Some tools, like `Any23`[7], inject data from different sources (CSV and HTML micro-formats) into RDF triples. However, these tools are not meant to discover the underlying structure of the data, validate it, or synthesize a full interactive application. Even though not specifically for open-data, the database community (specially in data warehouses) has proposed specialized Extraction-Transformation-Loading (ETL) tools to access and integrate data from different sources [16]. These tools deal with the problems of data cleaning [13] and data mapping to specific schemas. While some existing commercial tools offer advanced *generic* mapping tools[8], our proposal in this respect is the construction of *domain-specific* injectors equipped with heuristics to suggest relations between the data and automate cleansing through integrity constraints, which otherwise would need costly, expert guidance. In this line schema discovery approaches, like the one proposed in [8] for JSON documents, are of interest to facilitate the construction of injectors.

Other proposals [7] aim at documenting open data with metadata to describe its structure and ease the construction of applications for them. Our approach is compatible with those based on linked data, since transformations can convert the injected data into publishable RDF triples.

VII. CONCLUSIONS AND FUTURE WORK

We have presented a novel architecture for the engineering of Data Service applications, specially tailored for open-data applications, termed Open-Data-as-a-Service (ODaaS). The approach is based on multi-level modelling to represent domain knowledge. It supports on demand data loading through domain injectors, which are described through semantic-rich query descriptions. The data services are made available through a REST API, and we have shown a prototype implementation, including a generic web data exploration console. We believe an MDE-based approach for ODaaS applications is pertinent, as it makes it possible to assign semantics to data by promoting data as semantically rich models. Moreover, MDE also enables the generation of domain-specific applications for different platforms, like the web or mobile devices.

We are currently working on different aspects of the implementation. We plan to complete the framework for generating domain-specific applications on top of the Data Services using MDE. We are also interested in a deeper study of the integration of dynamic and static data sources [15]. Finally, we are also working on the interoperability of our frameworks with OWL ontologies, and RDF triplestores.

Acknowledgements. Work supported by the Spanish Ministry of Economy and Competitivity with project Go-Lite (TIN2011-24139) and the EU commission with project MONDO (FP7-ICT-2013-10, #611125).

[7] http://any23.apache.org/

[8] http://www.mulesoft.org/

REFERENCES

[1] C. Atkinson, B. Kennel, and B. Goß. Supporting constructive and exploratory modes of modeling in multi-level ontologies. In *SWESE*, pages 1–15, 2011.

[2] C. Atkinson and T. Kühne. The essence of multilevel metamodeling. In *UML*, volume 2185 of *LNCS*, pages 19–33. Springer, 2001.

[3] J. Bézivin and I. Kurtev. Model-based technology integration with the technical space concept. In *Proc. Metainformatics Symposium*. Springer, 2006.

[4] M. J. Carey, N. Onose, and M. Petropoulos. Data services. *Commun. ACM*, 55(6):86–97, 2012.

[5] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–646, 2006.

[6] J. de Lara and E. Guerra. Deep meta-modelling with METADEPTH. In *TOOLS'10*, volume 6141 of *LNCS*, pages 1–20. Springer, 2010.

[7] A. Gregory. Open data and metadata standards: Should we be satisfied with 'good enough'?, 2011. Open Data Foundation.

[8] J. L. C. Izquierdo and J. Cabot. Discovering implicit schemas in JSON data. In *ICWE*, volume 7977 of *LNCS*, pages 68–83. Springer, 2013.

[9] D. Lathrop and L. Ruma. *Open Government: Collaboration, Transparency, and Participation in Practice* . O'Reilly, 2010.

[10] OMG. MOF support for Semantic Structures (SMOF). `http://www.omg.org/spec/SMOF/`.

[11] OMG. MOF 2.0. `http://www.omg.org/spec/MOF/2.0/`, 2009.

[12] Open Data Commons. `http://opendatacommons.org/`.

[13] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[14] F. Schomm, F. Stahl, and G. Vossen. Marketplaces for data: an initial survey. *ACM SIGMOD Record*, 42(1):15–26, 2013.

[15] A. M. Segura, J. de Lara, and J. S. Cuadrado. Twiagle: a tool for engineering applications based on instant messaging over twitter. In *ICWE*, volume In Press of *LNCS*, pages 1–4. Springer, 2014.

[16] D. Skoutas, A. Simitsis, and T. K. Sellis. Ontology-driven conceptual design of ETL processes using graph transformations. *J. Data Semantics*, 13:120–146, 2009.

[17] The Open Data Protocol (OData). `http://www.odata.org/`.

[18] D. Varró. Model transformation by example. In *MoDELS*, volume 4199 of *LNCS*, pages 410–424. Springer, 2006.

[19] M. Völter and T. Stahl. *Model-driven software development*. Wiley, 2006.

[20] Q. H. Vu, T.-V. Pham, H.-L. Truong, S. Dustdar, and R. Asal. DEMODS: a description model for data-as-a-service. In *AINA*, pages 605–612, 2012.

[21] W3C. `http://www.w3.org/TR/wsdl`, 2001.