An Interoperability Tool for Low-Code Development Platforms

Rita Clode Silva Jardim Fernandes

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Miguel Leitão Bignolas Mira da Silva

Examination Committee

Chairperson: Prof. David Manuel Martins de Matos
Supervisor: Prof. Miguel Leitão Bignolas Mira da Silva
Member of the Committee: Prof. Sérgio Luís Proença Duarte Guerreiro

October 2021

Acknowledgments

First and foremost, I would like to express my gratitude to Prof. Miguel Mira da Silva my supervisor for introducing me to this challenge back in April 2020 and for his guidance during this research.

I would also like to acknowledge the Phoenix team, who gave me the opportunity to be a part of this project and have taught me so much. In particular Leonardo for all the knowledge you shared with me, for your guidance and support over this last year.

A special thanks my parents and sister Leonor for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible. You helped me grow as a person.

I thank my friends, family and colleagues that have crossed paths with me, taught me something and had a positive impact in me.

Last but not least, to Dinis that has been there for me during the good and bad times in my life. Thank you.

To each and every one of you, thank you.

Abstract

The development of software systems commonly requires integration use cases, such as the data exchange between multiple tools. Interoperability is defined as the ability of multiple software intermediaries to exchange information so that a tool is able to handle the information generated by another one. Over the last few years, Low-Code Development Platforms (LCDPs) have gained popularity, with a rising number of companies using them to build enterprise-grade apps and transform their businesses. The lack of interoperability will raise a common problem, since applications are changing from thick clients to thin web clients. We create a method to expose an Open Data Protocol (OData) service dynamically from an LCDP application, in order to be further consumed by other systems such as Business Intelligence tools. OData is a protocol that allows web clients to publish, query, and update data in data services using simple HTTP requests. All the artifacts necessary to have an OData service up are generated from an LCDP application's data model, including the translation of OData requests to SQL queries and compliance with the OData protocol. Our approach creates an API exposing the data retrieved from the LCDP application. The model is exposed as an OData service, allowing end-users to obtain data using the OData query language in a simple way, and for the data to be consumed by other applications.

Keywords

Open Data Protocol (OData); Low-Code Development Platforms (LCDPs); Data Integration; OutSystems.

Resumo

O desenvolvimento de sistemas de software geralmente requer ferramentas de integração, como a integração de dados. A interoperabilidade é definida como a capacidade de múltiplos intermediários de software de trocar dados para que uma ferramenta seja capaz de lidar com os dados gerados por outra. A popularidade das plataformas de desenvolvimento de low-code aumentou nos últimos anos, com o número crescente de empresas usando-as para criar aplicações de nível empresarial. A falta de interoperabilidade levantará um problema comum. Nesta investigação criamos um método para expor um serviço Open Data Protocol (OData) dinamicamente a partir de uma aplicação de plataforma de desenvolvimento de low-code, para ser posteriormente consumido por outros sistemas, como ferramentas de Business Intelligence. OData é um protocolo HTTP que permite que clientes da Web publiquem, consultem e atualizem dados em serviços de dados usando pedidos HTTP simples. A partir de um modelo de dados inicial obtido a partir de uma aplicação low-code, derivamos todos os artefatos necessários para ter um serviço OData que está em conformidade com a definição do modelo, incluindo a conversão de pedidos OData em consultas SQL de acordo com o protocolo OData. A nossa abordagem cria uma API expondo os dados recolhidos de uma aplicação low-code. O modelo é exposto como um serviço OData, permitindo que os clientes utilizem a linguagem de consulta OData para obter as informações de que precisam de forma fácil, além de permitir que sejam consumidas por outras aplicações.

Palavras Chave

Open Data Protocol (OData); Plataformas de Desenvolvimento Low-Code; Integração de Dados; OutSystems.

Contents

1 Introduction **1**

1.1 Research Methodology 3

1.2 Document Outline 4

2 Low-Code Development Platforms **7**

2.1 Architecture and Main Components of Low-Code Development Platforms 9

2.2 Development process in Low-Code Development Platforms 11

3 Systematic Literature Review **13**

3.1 Planning the Review 15

3.1.1 Motivation 15

3.1.1.A Open Data Protocol 16

3.1.2 Research Questions 17

3.1.3 Search String and Data Sources 17

3.1.4 Study Selection Criteria 17

3.1.5 Data Extraction 18

3.1.6 Data Synthesis 18

3.2 Conducting the Review 19

3.2.1 Results 19

3.3 Reporting the Review 20

3.3.1 RQ1. How does OData work? 21

3.3.2 RQ2. What are the main benefits and utility of using OData? 24

3.3.3 RQ3. What are the main challenges and limitations of the OData protocol? 25

3.3.4 RQ4. What applications have been developed with OData? 26

3.4 Lessons Learned 29

4 Research Problem **31**

5 Proposal **35**

5.1 Objectives 37

5.2 Description 37

5.2.1 Creation of OData’s EDM 38
5.2.2 Mapping between OData requests and SQL statements 39
5.2.3 De/Serialization Process 41

6 Demonstration 43

6.1 Context 45
6.2 Exposing an OData service in OutSystems 45
6.2.1 Creation of OData’s EDM 45
6.2.2 Mapping between OData requests and SQL statements 47
6.2.3 De/Serialization Process 48
6.3 OutSystems CDS Project 49
6.3.1 Example 51

7 Evaluation 53

7.1 Test Application 55
7.2 Unit Tests 56
7.2.1 Service and Metadata Documents 56
7.2.2 CRUD Operations 56
7.2.3 Querying Requests 58
7.2.4 PowerBI Integration 61

8 Conclusion 63

8.1 Research Contributions 65
8.2 Research Limitations 65
8.3 Future Work 65

Bibliography 67

A SLR Obtained Studies Table 73

B An OData Service Document Listing 75

C An OData Metadata Document Listing 77

List of Figures

1.1 DSRM process with the research context, adapted from [1]. 4

2.1 Layered architecture of LCDPs adapted from [2]. 10
2.2 Main components of LCDPs adapted from [2]. 10

3.1 Study selection process. 19
3.2 Distribution of the selected documents per dataset. 20
3.3 Distribution of the selected documents per year. 20
3.4 URI components of the Open Data Protocol. 23

5.1 Integration process using OData. 38
5.2 Example of a Product collection. 42

6.1 Entity diagram of CDS Project. 50
6.2 URL components of published and draft projects. 51
6.3 UI of the Overview screen when creating a new project. 51
6.4 UI of the Entities screen when creating a new project. 52
6.5 UI of the Security screen when creating a new project. 52

7.1 The Northwind DB entity diagram 55
7.2 POST request and response of creating a new entity in Shipper collection, using Postman. 57
7.3 PUT request for updating a specific entity in Shipper collection, using Postman. 57
7.4 Getting OData feed data in a PowerBI report. 61
7.5 PowerBI report with the project’s OData feed loaded data. 62

List of Tables

3.1 Inclusion and exclusion criteria. 18
3.2 Data extraction form. 18
3.3 Description of HTTP requests in OData. 23
3.4 Most used query options. 24

5.1 Integration process of an OData producer (LCDP application) and consumer. 38
5.2 Transformation of the target resource path into the corresponding SQL statement, through a query model. 40
5.3 Mapping query options into SQL statements, through a query model. 41

A.1 List of the obtained studies. 74

Listings

6.1 Example of an OData GET request mapping to SQL statement. 47
6.2 Example of an OData DELETE request mapping to SQL statement. 47
6.3 Example of an OData POST request mapping to SQL statement. 48
6.4 Example of an OData PUT request mapping to SQL statement. 48
6.5 Pseudo-code of OData requests execution. 49
7.1 GET request on Shipper entity set. 56
7.2 Shipper single entity with id=4. 57
7.3 Shipper single entity with id=4, after updating the company’s name. 58
7.4 Two of the top 3 most expensive beverages. 59
7.5 Checking the category of the product belonging to order detail with id=2. 59
7.6 Products with Chef names that aren’t discontinued, with the respective categories. . . . 60
7.7 Product out of stock awaiting order arrivals. 60
7.8 Inline count and second cheapest product query result. 61
B.1 Metadata document of the Northwind project. 76
C.1 Metadata document of the Northwind project. 77

Acronyms

ACM	Association for Computing Machinery
API	Application Program Interface
BI	Business Intelligence
BPMN	Business Process Model and Notation
CDS	Common Data Service
CoAP	Constraint Application Protocol
CRM	Customer Relationship Management
CRUD	Create, Read, Update and Delete
CS	Computer Science
CSDL	Conceptual Schema Definition Language
DB	Database
DSRM	Design Science Research Methodology
EDM	Entity Data Model
ERP	Enterprise Resource Planning
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IS	Information Systems
IT	Information Technology
JSON	JavaScript Object Notation
LCDP	Low-Code Development Platform
OData	Open Data Protocol
OGDI	Open Government Data Initiative
PaaS	Platform as a Service

REST	Representational State Transfer
RQ	Research Question
SCM	Supply Chain Management
SDK	Software Development Kit
SE	Software Engineering
SLR	Systematic Literature Review
SOS	Sensor Observation Service
SQL	Structured Query Language
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VGT	Virtual Global Taskforce
XML	Extensible Markup Language

1 Introduction

Contents

1.1 Research Methodology 3

1.2 Document Outline 4

The development of software systems commonly requires integration use cases, such as the data exchange between multiple tools. Interoperability is defined as the ability of multiple software intermediaries to exchange information so that a tool is able to handle the information generated by another. Since the representation of transferred data differs between tools, implementing an interoperability solution frequently calls for the use of syntactic and semantic mapping.

Low-code is a form of software development in which applications and processes are developed with little to no coding. A Low-Code Development Platform (LCDP) employs visual interfaces through simple logic and graphical features instead of sophisticated code languages. These platforms have become more popular as a cost-effective and time-saving alternative to traditional software development [3].

However, there is a lack of advanced functionalities within the LCDPs resulting in a need for interaction with other tools, such as data analytics and visualization tools. Such interaction frequently involves exposing web services, through Application Program Interfaces (APIs), tools that help support interoperability. OutSystems [4], one of the leading LCDPs has been trying to make their applications interoperable for almost two decades.

This research intends to create a dynamic interoperability tool for Low-Code Development Platforms using the Open Data Protocol (OData), a protocol which is further explained in Chapter 3. Such method will allow complex queries against the exposed information, enable Create, Read, Update and Delete (CRUD) operations over the service data, provide the means to navigate through relationships between entities and ensure the OData service can be consumed by an OData consumer.

1.1 Research Methodology

The research methodology adopted in this research is Design Science Research Methodology (DSRM).

DSRM generates and assesses IT artifacts such as constructs, models, methods, instantiations, or any created item with an embedded solution, to an understood research topic in order to tackle recognized research challenges [1]. It entails a rigorous technique for creating artifacts that follows a six-step procedure:

- **Problem identification and motivation** involves outlining the research challenge and justifying the solution's usefulness in order to encourage the researcher to pursue it.

- **Definition of the objectives for a solution** includes inferring the solution's objectives, either quantitative or qualitative, from the problem statement and knowledge about the problem's current condition and potential solutions.

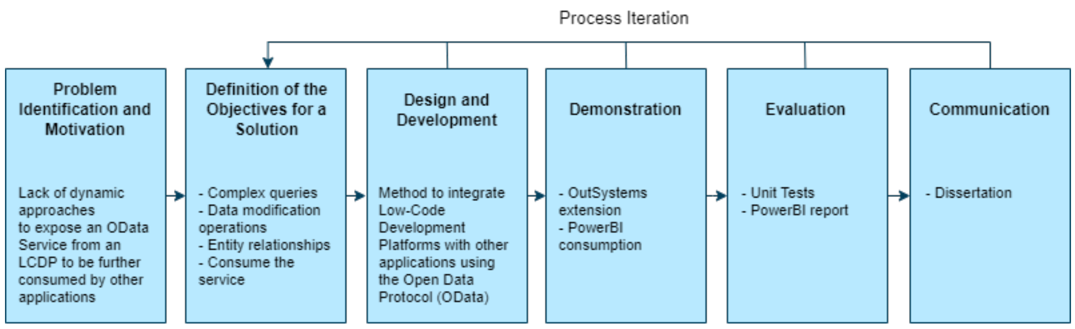

Process Iteration
Problem Identification and Motivation
Lack of dynamic approaches to expose an OData Service from an LCDP to be further consumed by other applications
Definition of the Objectives for a Solution
- Complex queries
- Data modification operations
- Entity relationships
- Consume the service
Design and Development
Method to integrate Low-Code Development Platforms with other applications using the Open Data Protocol (OData)
Demonstration
- OutSystems extension
- PowerBI consumption
Evaluation
- Unit Tests
- PowerBI report
Communication
- Dissertation

Figure 1.1: DSRM process with the research context, adapted from [1].

- **Design and development** entails the process of determining the artifact's desired functionality and architecture, then building it.
- **Demonstration** implies demonstrating how the artifact is applied to resolve at least one instance of the problem.
- **Evaluation** includes observing and measuring how effectively the artifact supports a solution to the problem, by contrasting a solution's objectives to real results from the demonstration's use of the artifact.
- **Communication** involves informing researchers and other relevant audiences about the topic and its importance, as well as the artifact's utility, novelty, and effectiveness.

This process, summarized in Fig. 1.1, is usually an iterative one. Researchers determine whether to try to increase the effectiveness of the artifact by redesigning and redeveloping it or to move on to communication and leave further improvement to future projects after analyzing the artifact [1].

This research was conducted using the DSRM [1] guiding principles, practice guidelines, and procedure for artifact deployment and evaluation.

1.2 Document Outline

The remainder of this document is structured as follows. Chapter 2 concerns Low-Code Development Platforms and how they work as our research field. OData is further described in Chapter 3 through a Systematic Literature Review (SLR) composed by the planning (Section 3.1), the conducting (Section 3.2), and the reporting phases (Section 3.3) with an analysis of the obtained results (Section 3.4).

The Research Problem of this master's thesis is presented in Chapter 4. A method for the dynamic integration process between an LCDP and an OData consumer through an OData service, which is our proposal to mitigate the defined problem, is developed in Chapter 5.

The demonstration of the integration process is covered in Chapter 6, and its evaluation is done in Chapter 7.

Finally, Chapter 8 exposes the contributions of our research, our main limitations and our intentions for future work.

2

Low-Code Development Platforms

Contents

2.1 Architecture and Main Components of Low-Code Development Platforms 9

2.2 Development process in Low-Code Development Platforms 11

In this chapter, we introduce Low-Code Development Platforms as our main research field.

Low-Code Development Platforms are simple visual environments that are increasingly introduced and promoted by major IT companies [2]. Such platforms help dealing with the shortage of highly-skilled software developers by enabling end users, with little to no programming experience, to contribute in software development processes. The most representative LCDPs are OutSystems [4], Mendix [5], Appian [6] and Kissflow [7].

LCDPs allow the development and deployment of fully functional software applications using powerful graphical User Interfaces (UIs) and visual abstractions requiring minimal or no procedural code [8]. They are frequently delivered on the cloud via a Platform as a Service (PaaS) model. PaaS is a cloud development and deployment environment that contains tools for building everything from simple cloud-based applications to sophisticated enterprise software enabled through the cloud [9]. PaaS help avoiding the cost and complexity of purchasing software licenses, development tools, managing application infrastructure and other resources. Model-driven engineering techniques are used to design these fully functional applications, which take advantage of cloud infrastructures, automatic code generation and graphical abstractions. To ensure effective and efficient development, PaaS models are used alongside deployment and maintenance, and software design patterns and architectures.

2.1 Architecture and Main Components of Low-Code Development Platforms

According to Sahay et al. [2], from an architectural point of view LCDPs are composed by four main layers, overviewed in Fig. 2.1:

1. **Application Layer**: The top layer consists of the graphical environment that users directly interact with, along with the toolboxes and widgets used to build the UI of an application. The authentication and authorisation mechanisms are also defined in this layer. Users also specify the behaviour of the application being developed.

2. **Service Integration Layer**: This layer is used to connect with different services through APIs and authentication mechanisms.

3. **Data Integration Layer**: The data integration layer is concerned with data integration from different data sources, allowing the data to be operated and homogeneously manipulated, even if heterogeneous sources are involved.

4. **Deployment Layer**: The developed application can be deployed on dedicated cloud infrastructures or on-premise environments, depending on which LCDP is being used.

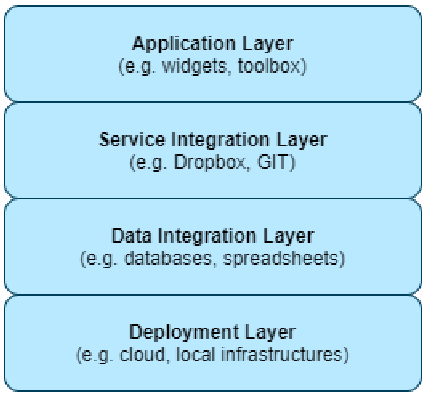


Figure 2.1: Layered architecture of LCDPs adapted from [2].

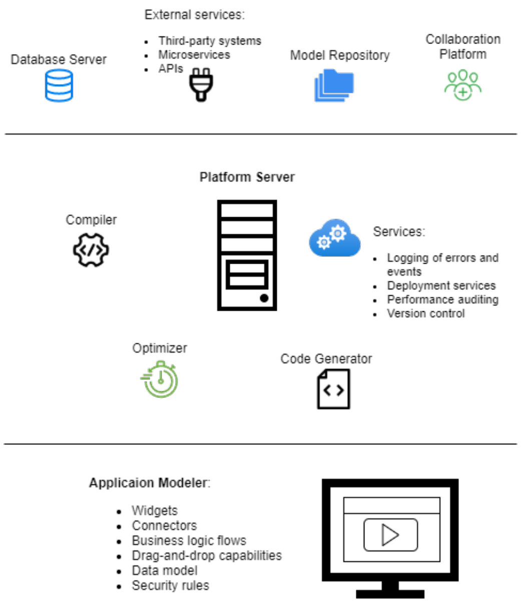


Figure 2.2: Main components of LCDPs adapted from [2].

The individual components that make up any LCDP are represented in Fig. 2.2 and through the expansion of the layered architecture defined in Fig. 2.1 they are divided into three tiers. The application modeler is in the first tier, the server side and its many features are in the second tier, and external services that are integrated with the platform are in the third layer.

The application modeler allows to specify applications using modeling constructs and abstractions. Once the application model is complete, it may be transferred to the platform back-end for further analysis and manipulations, including the construction of a fully functional application that has been tested and is ready to be deployed on the cloud. The middle tier takes the application model received from the application modeler and performs model management operations such as code generation and optimizations while also taking into account the involved services including database systems, micro-services, API connectors, model repositories of reusable artifacts, and collaboration means.

The developer is not concerned about the application's low-level architecture. All of the required micro-services are established, orchestrated and managed in the back-end. Developers are also relived from handling technical aspects manually such as business logic consistency, authentication, data integrity, load balance and security.

LCDPs handle version control by providing developers with repositories capable of storing reusable modeling artifacts. These platforms also offer capabilities that support development paradigms such as agile, scrum and kanban. As a result, developers can quickly visualize the application development process, define new tasks and sprints, deal with changes as soon as customers need them and engage with other stakeholders.

2.2 Development process in Low-Code Development Platforms

The typical phases involved in designing applications with LCDPs according to Sahay et al. [2] are:

1. **Data modeling** - through visual interfaces users configure the data schema of the application under development; such step includes generating entities, establishing associations, defining constraints and dependencies and so on;

2. **User interface definition** - users configure forms and pages in order to construct application views, followed by defining and managing user role and security mechanisms; drag-and-drop features help speed up development and render multiple views rapidly;

3. **Business logic rules and workflows specification** - users usually manage workflows among several forms or pages that require different operations on the interface components; Business Process Model and Notation (BPMN)-like notations can be used to implement such processes for visual-based workflows;

4. **External services integration** - LCDPs offer the consumption of external services by integrating various third-party APIs;

5. **Application deployment** - most platforms allow users to easily preview and deploy the produced application with only a few clicks;

3

Systematic Literature Review

Contents

3.1 Planning the Review 15
3.2 Conducting the Review 19
3.3 Reporting the Review 20
3.4 Lessons Learned 29

In this chapter we perform a Systematic Literature Review on the Open Data Protocol.

A Systematic Literature Review is a strategy for identifying, assessing, and interpreting all relevant research on a certain research issue or topic [10]. Systematic reviews attempt to give an unbiased and reproducible appraisal of a research topic using a well-defined, trustworthy, rigorous, and auditable process [10].

An SLR was conducted in order to summarize the existing information about the Open Data Protocol, to identify any gaps in current research and to offer a background for effectively placing new research activities relating to this topic. We performed our systematic research methodology guided by Kitchenham's Guidelines for performing Systematic Literature Reviews in Software Engineering [10]. The process of an SLR can be summarised into a three phase approach:

1. **Planning the Review**: Verifying the necessity for carrying out a review is done in this first phase. The major actions carried out here are identifying the research question(s) that the review is going to tackle and creating a review protocol, describing the core techniques.

2. **Conducting the Review**: We create and execute a search strategy in this phase. The actual relevance of the possibly relevant primary studies is assessed after they have been chosen. A data extraction is performed to record the information collected from the primary investigations.

3. **Reporting the Review**: The outcomes of a systematic review must be written up in the last phase. We also evaluate the data and explain the generalizability of the findings as well as the review's limitations.

3.1 Planning the Review

3.1.1 Motivation

As of today, the results delivered by several systems and organizations usually have little possibility of access, interaction and analysis of the data. Access to enterprise data from both external and internal applications is required by business practices. Suppliers, for example, make their inventories available to retailers, while health providers provide patients access to their medical records [11]. Data is frequently a scarce resource for small enterprises [12]. To see through this shortage of data sharing principles, opening the data would allow automatic data integration in application by third party users.

Many organizations plan to make a variety of data and services available to all or most client devices [13]. When a new client platform is installed, data and services must typically be updated and modified to accommodate the new platform. Similarly, when a new service is introduced, all existing client platforms must be updated to accommodate it [13].

With the growing availability of data in all domains, a common standard for data description is lacking. Thus, data sharing initiatives should be based on a common standard allowing interoperability. For example, opening existing surveillance systems in the public health domain could allow the development of innovative use of data and reduce the underuse of data, benefiting the public.

As the market trends towards easy access to data across different platforms and devices, this standard approach of exposing and consuming data is required in order to develop a more open and programmable web. Data services are a sort of web service that provide rich metadata, expressive languages, and APIs for querying and receiving data from service providers to customers [11]. The two most common approaches for consuming data services are through functions and queries [11]. Functions encapsulate data and restrict access to a set of carefully defined, usually application-specific function signatures. Queries, on the other hand, can be written in a supported query language based on the external model.

Companies are exposing their back-end business services all the more as plain old Hypertext Transfer Protocol (HTTP) endpoints. For the time being, Representational State Transfer (REST) has become the main protocol since web APIs are created depending only on Uniform Resource Identifiers (URIs) and HTTP messages. REST APIs boost data-driven applications that integrate data from different sources. Data-level integration refers to transferring, replicating, and transforming the data from one from one application to another, without regard for application or business logic.

3.1.1.A Open Data Protocol

The Open Data Protocol is a web service data access protocol that provides simple and standard building and consumption of queryable and interoperable RESTful APIs [14]. The service allows for the creation of data services, in which Uniform Resource Locator (URL)-accessible resources are specified using an Entity Data Model (EDM) and queried using conventional HTTP messages. EDM is an abstract data model that uses concepts like entity types, entities, and associations from the entity relationship model. The protocol also includes a URL-based query language as part of the URL format that resembles Structured Query Language (SQL) in some ways. This facilitates clients querying the data, through the uniform CRUD operations for the underlying data model. OASIS has accepted the latest version of OData (4.01) for standardization.

Even though some research has been done for merging and comparing data from different services, no SLR (Systematic Literature Review) has reviewed the literature regarding the use of the Open Data Protocol.

As a result, it's critical to gather all relevant studies in order to identify and comprehend the present state of OData research. It's therefore able to assess what concerns and questions have been addressed and solved, as well as what the most pressing issues with OData are right now.

3.1.2 Research Questions

The most crucial component of an SLR is defining the Research Questions (RQs), as they drive the entire review technique. This review intends to achieve two main objectives that are understanding the OData protocol and where is the protocol being used. The following RQs were established in order to meet the previously stated objectives:

- **RQ1.** How does OData work?
- **RQ2.** What are the main benefits and utility of using OData?
- **RQ3.** What are the main challenges and limitations of the OData protocol?
- **RQ4.** What applications have been developed with OData?

3.1.3 Search String and Data Sources

The Review Protocol is a detailed plan that outlines how a systematic review will be carried out [10]. It begins with a literature search, which outlines the specification of the search string that will be used to conduct a search across the selected datasets in order to find the greatest number of papers that may answer the research questions. The search string used to conduct the search, as well as the datasets that were selected, are listed below.

Search String. `"OData"` AND `"protocol"`

Datasets. EBSCO Host, Web Of Science, IEEE Xplore Digital Library, ACM Digital Library and Science Direct.

3.1.4 Study Selection Criteria

The study selection criteria of a systematic review are used to establish which research studies are excluded and which are included. Its goal is to find studies that are related to the research topics and provide direct proof. The review's inclusion and exclusion criteria are listed in Table 3.1.

Table 3.1: Inclusion and exclusion criteria.

Inclusion Criteria	Exclusion Criteria
- Written in English. - From one of these domains IT/CS/SE/IS. - Source type is academic journal, conference material, ebook/book, documentation, or journal. - Studies about the adoption of the OData protocol. - Studies identifying the risk factors associated with OData and impact on reliability.	- Not written in English. - Source type is patent. - Studies not related to the research questions.

Table 3.2: Data extraction form.

	Data Item	Description
1	Title	Title of the study
2	Authors	Name of the author(s)
3	Year	Publication year of the study
4	Publication type	Type of publication (e.g. conference/academic journal/book)
5	Data source	Which data source was the study retrieved from
6	Study aim	Aim of the study
7	Research focus	Which research questions the study relates with

3.1.5 Data Extraction

The data extraction process defines how the information required from each study will be obtained [10]. The purpose of this step is to create data extraction forms that correctly capture the information that researchers get from the gathered studies [10]. Thus, our data extraction form (Table 3.2) was designed. Data items 1 to 5 gathered basic information of the papers, including the title of the document, the name(s) of the author(s), the year of publication, publication type and respective data source which the document was retrieved from. After reading the publications, the remaining data items (6-7) were obtained. The extracted data items were collected, which aided in the data's organization and analysis.

3.1.6 Data Synthesis

Data synthesis entails collating and summarising the results of the final set of studies. The 37 selected studies were read noting recurrent methods and findings. Inconsistencies and discrepancies were recorded and are presented further on.

For RQ1, we extracted the information from studies which mentioned explicitly the components of OData and how this protocol works. Furthermore, we extracted the list of benefits, problems and challenges of using this technology to answer both RQ2 and RQ3. Regarding RQ4, we checked

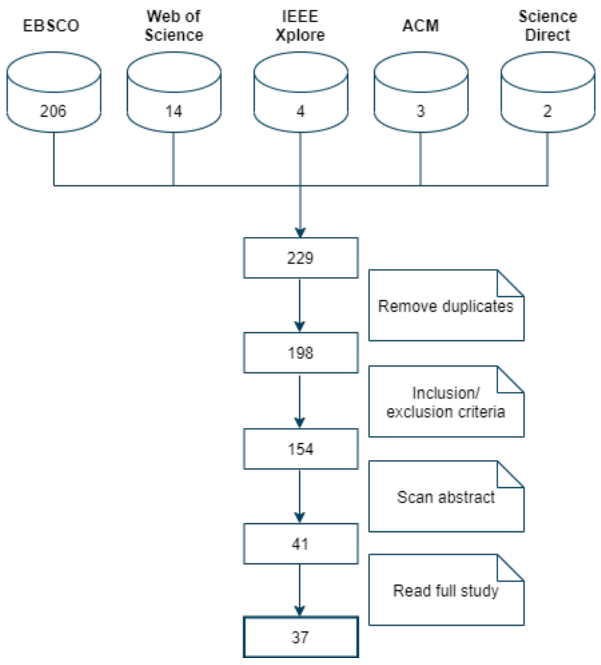


Figure 3.1: Study selection process.

whenever the studies would introduce a domain or a new tool that made use of the OData protocol.

3.2 Conducting the Review

The second stage of the SLR is undertaken in this section. The purpose is to use an objective search approach to discover as many primary studies that relate to the research topic. The designated search strategy is performed and the extracted data is analyzed.

The search string was applied to the Review Protocol's specified resources, yielding a total of 229 publications. The duplicates were removed and we were left with 198. After applying inclusion/exclusion criteria, 154 studies remained. Once the potentially relevant primary studies were obtained, the abstracts were read to further exploit the importance of the documents. For our final selection, we had a total of 37 studies. Fig. 3.1 presents the whole SLR execution process. Table A.1 contains the whole list of selected documents as well as some of the retrieved data items.

3.2.1 Results

In this section we examine at several aspects of the chosen research, such as their data sources and dispersion through time.

Among the 37 selected documents, EBSCO Host is the most represented source followed by Web of Science, as illustrated in Fig. 3.2. EBSCO host was expected to be the main source as it is one

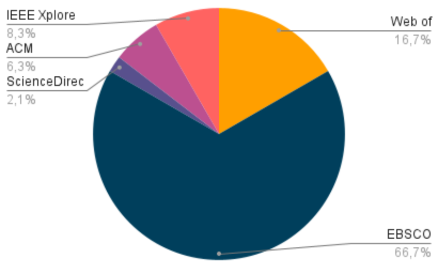


Figure 3.2: Distribution of the selected documents per dataset.

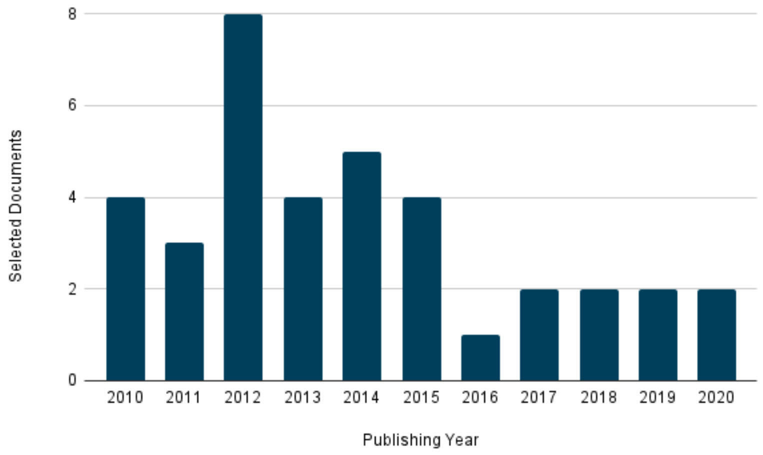


Figure 3.3: Distribution of the selected documents per year.

of the largest multidisciplinary databases for academic web resources in the world, with over 36 disciplinary databases.

Regarding the distribution over the years of the selected studies, by analyzing Fig. 3.3 we can check that 2012 has most relevant information related to our research. OData was created by Microsoft in 2007. Versions 1.0, 2.0 and 3.0 were released under Microsoft Open Specification Promise. In 2012, the OASIS international consortium launched an initiative to standardize OData. Releasing in 2014, OData's version 4.0 was already standardized at OASIS. Although this technology has been around for almost 15 years, few researchers have embraced the protocol in their work.

3.3 Reporting the Review

The last phase of a systematic review entails reporting the findings, responding to our four research questions, and communicating them to anyone who might be interested.

3.3.1 RQ1. How does OData work?

As already stated, Microsoft presented the Open Data Protocol back in 2007. By 2012, OData had been proposed to OASIS, and in 2014 Version 4.0 was released by the international open standard consortium. As of 2020, OData Version 4.01 has been published, which is a highly compatible, incremental release over OData 4.0.

OData is an open protocol on top of HTTP that allows web clients to use basic HTTP queries to publish, query, and update information in data services [15]. It enables you to develop resources that are specified by an Entity Data Model and queryable by web clients through a SQL-like URL-based query language [16]. This query language has a range of query options that allow customers to exactly define the instance data they want. Simply described, OData is a standardized data transport format with a defined data access interface [13]. The data is serialized and sent via HTTP using the Extensible Markup Language (XML) or JavaScript Object Notation (JSON) standards. The latter provide an alternative data format, which is supported in just about all web application technologies. The OData client ecosystem has grown over the previous few years to the point that client libraries are available for the main client devices and platforms, with more on the way [13]. The OData ecosystem is composed of service producers and service consumers. OData service producers use the OData protocol to expose their data, whereas OData consumers are simply applications that consume data exposed using the OData protocol. OData consumers can range in sophistication from a simple web browser to a custom application that exploits all of OData's features. *API Sever*, *BrightstarDB*, *IBM App Connect*, *Lightswitch* and *Windows Azure Table* Storage are some of the many OData producers.

OData consists of the following four main parts [17]:

- OData protocol - The protocol specifies the way consumers can interact with data sources. CRUD operations along with the supported XML and JSON serialization standards. The query language includes a set of query parameters that enable customers to describe the data they want.

- OData data model - An abstract data model, the EDM, defines the data structure and provides a general mechanism to detail and arrange the data. It's an instance of an entity relationship model implementation, in which data is represented as entities and relationships between them. A Service Metadata Document is provided by an OData service, and it describes the service's EDM-based model in the XML-based Conceptual Schema Definition Language (CSDL).

- OData service - An OData service exposes a callable endpoint that is used for accessing data or calling functions. It employs the data model, implementing the OData protocol.

- OData client - An OData client uses the OData protocol and the corresponding OData data model to connect to an OData service.

The service document lists all the top-level feeds for users to access them, since a service may contain one or more feeds [15]. It helps service consumers to find the locations of the available resource collections, since the document lists the collections of available resources provided by the service. The service document is returned when making a get request on the service root URI. The service metadata document specifies its Entity Data Model, through the “$metadata” request [15]. The OData metadata document is the standard way to let end-users know how to query the data, as it presents information about the structure and organization of all the resources. The result is in CSDL format.

Bellow we list the main concepts in the EDM:

- Entities are instances of entity types, such as Product or Category.
- Entity types are structured types with a name and a key [14]. They specify the entity’s properties and relationships. The key of an entity type is made up of a subset of the respective primitive properties, such as ProductId or CategoryId.
- Navigation properties are used to represent relationships between entities. They are usually defined as part of an entity type [18]. There is a specific cardinality to each relationship.
- Complex types are structured types with a name but no key, composed of a group of properties. They can’t be referred to outside of the entity that holds them. Usually addresses are represented as a complex type.
- Entity sets are collections of entities with a specific name, for example Products is an entity set containing Product entities. An entity’s key is used to identify it inside an entity set [18].
- Operations allow the execution of custom logic to be run on sections of the data model [18]. Functions and actions are operations. The main difference between them is that functions don’t have side effects and allow further composition while actions allow side effects (such as data modification) but cannot be further composed.

OData uses HTTP verbs (GET, PUT, POST, DELETE) to define actions on resources, and it uses a common URI syntax to identify those resources. The client must perform an HTTP POST, GET, PUT, or DELETE request to create, read, update, or delete an object, accordingly [11]. The HTTP requests are summarized in Table 3.3.

Table 3.3: Description of HTTP requests in OData.

HTTP Request	Description
GET serviceRoot/Products	Returns the collection of Products.
DELETE serviceRoot/Products(3)	Deletes the Product with ProductId = 3.

PUT serviceRoot/Products(3)
{
"odata.type" : "#.Northwind.Product",
"ProductName" : "Chai"
}
	Updates the ProductName of the Product with ProductId = 3 to "Chai"

POST serviceRoot/Product
{
"odata.type" : "#.Northwind.Product",
"ProductId": 6,
"ProductName": "Chai",
"CategoryId": 1,
}
	Creates a new Product with the respective details

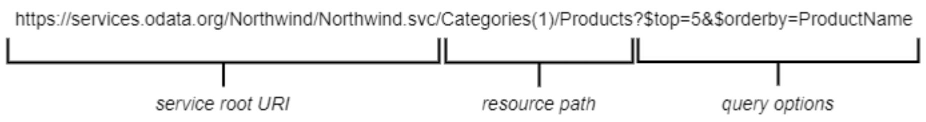

https://services.odata.org/Northwind/Northwind.svc/Categories(1)/Products?$top=5&$orderby=ProductName
service root URI
resource path
query options

Figure 3.4: URI components of the Open Data Protocol.

OData also defines a set of rules for producing URIs to identify the data and information given by an OData service [15]. The service root URI, the resource path, and the query options are the three main URI components, which are displayed in Fig. 3.4. The root of an OData service is identified by the service root URI [17].

The resource path specifies the resource with which the service consumers want to interact. It is mostly used to address a collection, an entity within a collection, an entity's attribute or a relationship. To address a collection, the resource path is simply the name of that collection [15]; for example, `https://myhost/Northwind.svc/Products` would address the set of entities in the Products entity set. To address an entity within a collection, the resource path is the name of the collection followed by a key predicate in brace marks [15]. The URI `https://myhost/Northwind.svc/Products(2)` would address the product with ID = 2. To address a property of an entity, we must first find the entity and then determine the property's name [15]; for example, `https://myhost/Northwind.svc/Products(1)/Name` would retrieve the attribute Name of the product with ID=1. To address a relationship between entities, we must attach the name of the relationship to the end of the URI `https://myhost/Northwind.svc/Products(2)/Category`. It is possible to navigate through several tiers of relationships using this syntax [19].

Query options in the URL request allow you to influence how the service processes a request. To customize a request, OData offers a set of system query options. System query options are prefixed with the $ character (optional in OData 4.01). The most used query options are explained

Query Option	**Description**
$top=n	The service returns the number of items that are available up to and including the supplied value n.
$skip=n	The service returns items starting at position n+1
$orderby=PropertyName	Specifies the property to order by the items returned from the service.
$count=true	Indicates that the total number of items in a collection that matches the request should be delivered with the result.
$filter=PropertyName eq Value	Restricts the set of items returned over one or more specified properties.
$select=PropertyName	Only the properties, functions, dynamic properties and actions that have been specifically requested should be returned by the service.
$search=SearchExpression	Only the items that match the supplied search query are included in the result.
$expand=RelatedEntity	Indicates the related entities that must be represented inline.

Table 3.4: Most used query options.

in Table 3.4.

For an OData service, the protocol specifies three conformance levels: minimal, intermediate and advanced [14]. Each level includes a collection of requirements and suggestions that must be met by a service in order to fulfill the level's standards [14].

3.3.2 RQ2. What are the main benefits and utility of using OData?

OData is a big step forward in open interoperability standards, allowing users to retrieve and update data across different platforms [20]. The same data can be consumed by multiple client systems, ranging from phones to computers. Data from different sources such as file systems, traditional websites, relational databases and content management systems can be retrieved via OData, enabling developers to build cross-platform web and mobile applications, facilitating connection and information exchange across implementations [21].

Integration and interoperability are essential goals, but OData also allows data access. By using modern and well-established web standards, OData provides refreshingly simple data access. It has become the common choice for publishing datasets online due to its ease of use for both client applications and end-users. In addition, client code is easy to maintain and deploy [22]. Developers and web service architects will be comfortable with OData, since it is based on established standards and familiar design methodologies. Moreover, because OData feeds are accessed via HTTP, they can be displayed in any browser.

Service providers can never predict what kinds of queries/services will be requested by consumers. They would always have to provide new services or add parameters in order to meet the client's needs. OData can help solve this problem, by enabling flexible querying. The protocol enables flexible querying and filtering on all attributes, and this flexibility is critical when dealing with larger amounts of data. The ability to project a subset of properties minimizes both the size of the HTTP responses and the memory footprint of the client-side objects [19]. By updating the back-end service to expose an OData endpoint, each client gains the flexibility of downloading only the fields in which they are interested in.

OData has already been widely accepted by several companies due to its simplicity. It has a strong ecosystem with abundant resources including:

- consumers
- applications exposing OData
- live services (producers)
- sample services

This standard is also supported by libraries and toolkits, many of which are open source for a variety of platforms, making the construction of OData services easier.

OData also offers a rigorous specification for enabling new sources of data connectivity, despite where or how such sources are connected. This data source access mechanism is a viable solution for data services with a few dozen to a few hundred items and throughput up to a few hundred samples per second [22].

3.3.3 RQ3. What are the main challenges and limitations of the OData protocol?

In order to create an OData service, we need to [14]:

1. represent the data models in OData format
2. implement a converter that accepts OData requests and transforms them into SQL statements or the target storage technology of choice
3. serialize/deserialize messages in accordance with the OData protocol

Thus creating OData services is a time-consuming and tedious task for data providers.

In today's networking world, no data acquisition method can be safely used without security, which includes authenticating the user and, in certain cases, encryption [22]. OData by itself is not secured in a way that feels acceptable, especially being an open data initiative. To ensure the data exposed is secured properly, the client must add an additional layer of protection.

OData is a protocol for service-oriented environments, hence it requires robust development and security governance. In addition to this, clients must also cope with communication failures and unreliable messages [23]. Because the server in REST is unable to retain any state, state management must be performed by the client as well [23].

3.3.4 RQ4. What applications have been developed with OData?

Sensor Observation Service (SOS) by the Open Geospatial Consortium offers standard web service protocols for securely transmitting sensor data online [15]. However, the SOS has a limited ecosystem that makes it difficult to create and consume, supporting only predefined queries. A sensor data mediator solution was proposed by Huang et al. [15] to establish an SOS Entity Data Model for OData to bridge these two standards. The proposed system is able to covert between SOS and OData services with ease. The transformed services can be used directly by both SOS and OData clients that are compatible with the standards.

Ed-Douibi et al. [16] developed a model-based solution to automatically compose and orchestrate data-driven REST APIs. This method creates a global API from a set of initial REST APIs that are expressed as Unified Modeling Language (UML) models, exposing a single data model that combines all data models of the original APIs [16]. The global model is provided as an OData service, allowing users to quickly conduct queries using the protocol's query language. Queries on the global model are converted into queries on the many APIs that make up the underlying infrastructure, dynamically.

The creation and implementation of a framework for symmetric integration of applications and systems in cloud environments is proposed by Muntean et al. [24], with the goal of ensuring the completeness and integrity of data during the integration process. OData served as an integrator between SAP Hybris Cloud for Customer (C4C) and SAP S/4 HANA Cloud (S/4), for development and validation of the presented framework. OData is the main technology used in SAP's mobile and web applications, such as SAP Fiori apps, to access corporate data [25]. SAP NetWeaver Gateway technology, for instance, uses OData to expose SAP Business Suite software to customers on a variety of platforms [26].

Cardoso et al. [27] created a data model to store data generated by an electromyography, optimized for analytical processing. A web API was implemented to provide access to data in an agnostic way to database management systems and data consumers, using OData.

To make data from Enterprise Resource Planning (ERP) systems more accessible for integration with the Semantic Web, Kirchhoff et al. [28] developed an approach to implement SPARQL endpoints on top of OData interfaces. To answer a particular SPARQL query, the approach decides which OData services to be queried.

Although there are some Software Development Kits (SDKs) to develop OData applications, such as RESTier2, and commercial tools that expose OData services from existing data sources, like Cloud Drivers5, they still need sophisticated OData expertise to build the service's business logic, and they only support the OData standard to a limited extent [14]. Other programs, such as simple-odata-server8 and JayDATA9, can construct a basic OData server from an OData-formatted entity model and its associated database, but they only cover a subset of the protocol. Ed-Douibi et al. [14] derive all the artifacts required to get an OData service up and running on top of a relational database (that conforms to the model definition), from an initial UML class diagram.

To combat the proliferation of online child sexual abuse content, the Virtual Global Taskforce (VGT) suggested for OData to be adopted globally. The protocol would give much-needed interoperability across law enforcement tools in the worldwide community when it comes to dealing with criminal photographs and recordings [25].

In the public health domain, opening up current surveillance systems could benefit the public by enabling the development of creative data uses and minimizing data underuse due to restricted resources in research or surveillance teams. Using the Sentinelles system, which is a general population public health information system, a general purpose standard using OData was built by Turbelin et al. [29]. The OData web services exposing data and metadata are available at `http://odata.sentiweb.fr`.

When it comes to sensor motes, the Constraint Application Protocol (CoAP) is now the focus of both research and industry [17]. An embedded OData implementation on top of CoAP might be useful for easily integrating sensor motes into enterprise networks. Thoma et al. [17] provided this embedded OData implementation. The OData entity abstraction, which involves abstracting the topology and the sensors (or actors) as entities, is well suited to a typical sensor network application [17]. As a result, OData is a suitable abstraction for systems that can be thought of as sensor network databases.

SmartCampusAAU [30] is an open, extensible platform that makes building indoor location-based systems simple. It is designated to facilitate indoor positioning and navigation, offering an OData back-end that allows researchers to share radio map and location tracking data [30].

OData makes it ideal for presenting both near-real-time time-series streaming data and data from SQL back-end stores. Ross [22] demonstrates this with an implementation of an OData service-based industrial automation time-series data streaming capability.

OData is assisting in the transformation of Open Government projects to make government data available to the public. The cities of Regina and Medicine Hat in Canada, as well as the Colombian government, have developed open data catalogs based on the OData protocol [26]. Following the government's decision to make Met Office weather data available to the public, OData is making data more accessible to U.K. citizens. The Open Government Data Initiative (OGDI) is a Microsoft Windows Azure-based service that allows government bodies to publish a wide range of open data [19].

Moreover, the following applications also make use of OData:

- The back-end cloud service for the CLEO mobile sensing platform [31] uses the OData protocol to make data available to other systems, such as the World-Wide Telescope.
- The jQuery plug-in called DataTables that allows to slice and dice the large amounts of data, can consume data from publicly available OData services, as demonstrated by Lerman [32]
- EastBanc Technologies used OData to create a metro transit visualization application.
- Power Query, which is a popular plugin for Excel, has the ability to consume data from an OData API to render live workbooks stored in the Platform for Science.
- Viecore representatives explained how they used OData to construct advanced decision support and control systems for the US military, noting OData's flexibility and low client-server coupling as reasons for their ability to rapidly iterate on application designs and needs.
- IBM's WebSphere eXtreme Scale REST data service uses OData to provide access to the IBM eXtreme Scale data grid and this was done without the assistance of the OData team, due to the little technical friction.
- The PowerPivot add-in for Excel has an in-memory analysis engine that can be used directly from within the Excel interface and it allows importing data via an OData feed, integrating into data models that that also contain data from relational databases [13].
- WCF Data Services is a framework that automatically transforms an OData request into the underlying store's query language, based on the mapping between the EDM model and the relational model of the store.
- The OData Service Validation tool seeks to strengthen the existing OData ecosystem by allowing OData service developers to check their implementations against the OData standard to ensure that they work with any OData client.
- All lists and documents within those lists can be exposed as an OData feed in Microsoft SharePoint [19].

3.4 Lessons Learned

The foundation of modern business and industry is data. Data is collected and maintained in databases by applications, data is stored on the cloud by businesses and many companies make a living selling data. There are a plethora of data sources available and of potential clients: web browsers, mobile apps, and Business Intelligence (BI) tools are just a few. It makes far more sense to define a common methodology than exposing data sources in every possible way [23]. All that's required is agreement on how to model the data and a strategy for accessing to it.

Given our web-centric society, it would make sense to design this technology using existing web standards [23]. The Open Data Protocol is a data access protocol that allows developers to easily expose and access information from a variety of data sources such as relational databases, file systems, and content management systems via web services with query and update capabilities in a simple and standard way [14]. It offers a uniform and URI-based querying interface that maps CRUD operations to HTTP verbs.

OData makes it easy for analysts and developers to consume data from any platform or device by making it simple to interact with data from a wide range of apps and programming languages. It's the best solution to standardize the connection between applications and a wide range of enterprise data sources, allowing for better integration and interoperability between information providers and consumers [21].

Unfortunately, the protocol itself is not secured in a way that feels acceptable, especially being an open data initiative. To ensure the data exposed is secured properly, the client must add an additional layer of protection. Besides that, creating OData services is a time-consuming and tedious task for data providers.

However, being a standard, flexible interface allows the same API to be used across a number of clients. OData is the main technology used in SAP's mobile and web applications to access business data. It is also assisting in the transformation of Open Government projects to make government data available to the public.

This OASIS standard's long-term goal is to have an OData client library for every major technology, programming language, and platform, allowing any client app to read OData feeds [19]. As the OData specification evolves, new functionalities become available that service producers may seek to make available to service consumers.

4

Research Problem

According to the Design Science Research Methodology, this Chapter corresponds to the "Problem Identification and Motivation" step.

The paradigm of distributed Information Technology (IT) architectures is shifting away from monolithic programs running on a single node and moving towards distributed, dynamic environments [23]. Such environments enable the creation of applications by assembling existing services, increasing code reuse while reducing development time [23]. Ensuring the quality of data integration between systems and applications in these environments is essential [24]. Data integration refers to the transfer, replication, and transformation of data from one application to another without regard for application or business logic.

Over the last few years, Low-Code Development Platforms have gained popularity, with a rising number of companies using them to build enterprise-grade apps and transform their business. According to Gartner [33], low-code application development will account for more than 65% of all app development functions by 2024, with 66% of large companies adopting at least four low-code platforms. The lack of interoperability will raise a common problem, since applications are changing from thick clients to thin web clients [23].

Instead of implementing complex integration frameworks for every enterprise-integrated application (e.g., ERP, Customer Relationship Management (CRM), Supply Chain Management (SCM) systems), working towards a generic data integration method that allows interoperability among several applications should be our goal. This would allow interoperability between Low-Code Development Platforms and enterprise-integrated applications, Business Intelligence tools and other systems.

As an OASIS approved standard, OData is a viable alternative for open data exchange services [23]. Previous work has only focused on creating bridging applications or theoretical approaches for exposing OData services in a non automatic way.

Thus, the problem identified in this research is that there is a **lack of dynamic approaches to expose an OData Service from an LCDP to be further consumed by other applications**, without the support of an extra tool or a bridging application.

5 Proposal

Contents

5.1 Objectives 37
5.2 Description 37

This chapter defines the solution's objectives and explains our proposal in detail.

5.1 Objectives

The main goal related to this research is creating a method to expose an OData service dynamically from a Low-Code Development Platform application, in order to be further consumed by other systems (OData consumers) such as Business Intelligence tools. For this to be achieved we defined the following objectives:

- allow complex queries against the exposed information;
- enable CRUD operations over the service data;
- provide the means to navigate through relationships between entities;
- ensuring the OData service can be consumed by an OData consumer;

Trying to accomplish these objectives, we had in mind keeping the cost of our solution as low as possible, without the support of extra tools or a bridging applications.

5.2 Description

We generate all the artifacts required to have an OData service up and running from a data model retrieved from an LCDP application, through the translation of OData requests to SQL queries and compliance with the OData protocol [14]. Our approach creates an API exposing the data retrieved from the LCDP application. The model is exposed as an OData service, allowing end-users to obtain data using the OData query language in a simple way, and for the data to be consumed by other applications [16].

The integration process involves two independently designed applications, one of which is a Low-Code Development Platform in our approach. Data is retrieved from the LCDP and stored in the appropriate format in the Application Z illustrated in Fig. 5.1. Application Z stands for any application that has the ability to consume an OData service.

The actions listed in Table 5.1 illustrate the scenario for integrating an LCDP application (OData producer) with an OData consumer.

The OData service generation (A2) is the most complex action of the 3 listed. Developers should first specify their data models in the EDM format, then add business logic to resolve URLs using the OData query language to handle querying and modifying the data and finally translate such queries into SQL statements [14]. Furthermore, to exchange messages with OData clients who follow the

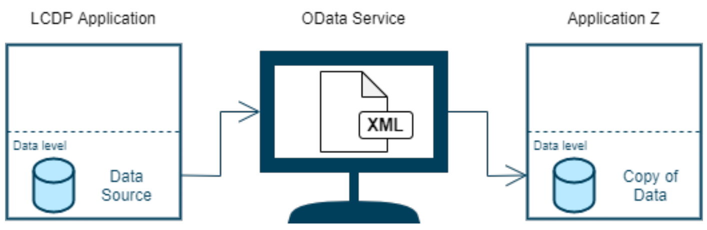


Figure 5.1: Integration process using OData.

Action Id	Action
A1	Retrieving table structures from the LCDP application's data model.
A2	Generating the OData service that exposes the tables retrieved.
A3	Consuming the generated data service through Application Z.

Table 5.1: Integration process of an OData producer (LCDP application) and consumer.

protocol, a de/serialization mechanism is necessary. Thus, the entire process of generating an OData service includes the following tasks:

1. The creation of the OData data model (EDM) from the received table structures.

2. The mapping between OData requests and SQL statements

3. The de/serialization process.

5.2.1 Creation of OData's EDM

The first step in defining an OData service is designing the entity model [22]. The main components of a database table are the table's name, primary key and a list of attributes which correspond to the table's columns. Each attribute has a name, a type, whether it is a primary key, whether it is foreign key referencing another table's primary key and, if so, a pointer to the corresponding primary key.

The Entity Data Model is a collection of concepts that describe the structure of data, regardless of how it is stored [18]. The entity type is the fundamental building block to express the structure of data. A group of instances of a particular entity type is referred to as an entity set. Each entity must have its own entity key within an entity set. Entity sets are all grouped in an entity container. Properties establish the structure and characteristics of entity types. A Product entity type, for example, might have properties like ProductId, Name and Price. A property can hold either primitive data (e.g., text, integer, or Boolean values) or structured data using complex types. Association types are used to describe relationships between two entity types. Every association has two association ends which correspond to the two related entity types and a multiplicity representing the maximum number of entities that can be at an association's end. A navigation property on an entity type is an optional property that allows users to navigate from one end of an association to the other end.

The mapping between a table structure and the corresponding EDM is the following:

- for each table an entity type is created, with the table’s name, primary key as the entity’s key;
- for each list of attributes from a table, properties are created within the corresponding entity, with the respective name and type;
- for each foreign key a navigation property is created, linking the source and target entities and is stored in the respective entity type;
- each record is placed within the corresponding entity set;
- an entity container is created to store the entity sets;

5.2.2 Mapping between OData requests and SQL statements

To transform OData requests to SQL statements we consider [14]:

- **HTTP method** - specifies if the request is either a query or a data modification action;
- **resource path** - identifies the resource to query or modify (e.g., products, a single product, supplier of a product);
- **query options** - allows to specify the required instance data

We created a query model to perform the transformation of the target resource path into the respective SQL statement with the specified query options. The model has the following structure:

- **list of output tables** - the tables to select from;
- **list of join references** - a join reference is composed of the table and the two columns to join on;
- **list of selected columns** - the columns that are selected from the output and join tables;
- **list of where conditions** - a where condition is composed of unary or binary expressions that contain the column name, the operator and the value (e.g., Name = ’John’); a where condition can have more than one expression by using the logical operators *OR* and *AND* (e.g., Name = ’John’ *AND* Age > 25);
- **list of orderby conditions** - an orderby condition is composed of the column name and the sort order (i.e., ascending or descending);
- **offset number** - the number of records to skip from the beginning;
- **next number** - the top *n* records to display;

Resource Path	Example	Model Parameters	SQL Statement
Collection of Entities	*Product*	- Add Product table to the list of output tables	SELECT * FROM Product
Single Entity within a Collection	*Product(2)*	- Add Product table to the list of output tables - Add Product's PrimaryKey = 2 to the list of where conditions	SELECT * FROM Product WHERE Product.Id = 2
Property of a Single Entity	*Product(2)/Name*	- Add Product table to the list of output tables - Add Product's PrimaryKey = 2 to the list of where conditions - Add Name column to the list of selected columns	SELECT Product.Name FROM Product WHERE Product.Id = 2
Relationship between Entities	*Product(2)/Category*	- Add Products table to the list of output tables - Add Category table to the join references with Category's primary key and Product's foreign key to Category as join columns - Add Product's PrimaryKey = 2 to the list of where conditions - Add all Category's columns to the list of selected columns.	SELECT Category.Id, Category.Name FROM Product JOIN Category ON Product.CategoryId = Category.Id WHERE Product.Id = 2

Table 5.2: Transformation of the target resource path into the corresponding SQL statement, through a query model.

- **count option** - a boolean value that indicates if the total number of records should be displayed;

We designed a method to perform the transformation of the target resource path into the corresponding SQL statement, which is detailed in Table 5.2. To query a collection of entities, the name of the collection is added to the list of output tables. Now, to get a specific entity within a collection besides adding the name of the collection to the list of output tables, one must specify a where condition of the primary key stated. A property of a particular entity is retrieved the same way as a single entity with the name of the property listed in the selected columns. Similarly, to navigate through a relationship of a specified entity we add the attributes and table as a joining reference and only specify the end of the relationship's columns.

OData requests are refined through query options, so we also had to take them into account in the mapping of the SQL statement. Table 5.3 describes which model parameters are defined and the corresponding SQL statement for each query option. Query options *top* and *skip* specify the number of records to be included and excluded in the request result through the *offset* and *next* numbers. To order the OData payload a particular column and the sort order are added as order by conditions. If no sort order is specified, our method assumes an ascending order, to comply with the OData

Query Option	Model Parameters	SQL Statement
$top=5	- Assign 5 to next number	SELECT * FROM Product FETCH FIRST 5 ROWS
$skip=3	- Assign 3 to offset number - Add Product's PrimaryKey = 2 to the list of where conditions	SELECT * FROM Product OFFSET 3 ROWS
$orderby=Price desc	- Add Price column and descending to the list of orderby conditions	SELECT * FROM Product ORDERBY Product.Price DESC
$count=true	- Assign true to count option along with the result	SELECT COUNT(*) FROM Product ——— SELECT * FROM Product
$filter=Price<10	- Add Price < 10 to the list of where conditions	SELECT * FROM Product WHERE Product.Price < 10
$select=Price	- Add Price column to the list of selected columns	SELECT Product.Price FROM Product
$search=Chai	- Add column like 'Chai' to the list of where conditions for every column in Product's table	SELECT * FROM Product WHERE Product.Id like 'Chai' OR Product.Name like 'Chai' OR Product.Price like 'Chai' OR Product.CategoryId like 'Chai'
$expand=Category	- Add Category table to the join references with Category's primary key and Product's foreign key to Category as join columns	SELECT * FROM Product JOIN Category ON Product.CategoryId = Category.Id

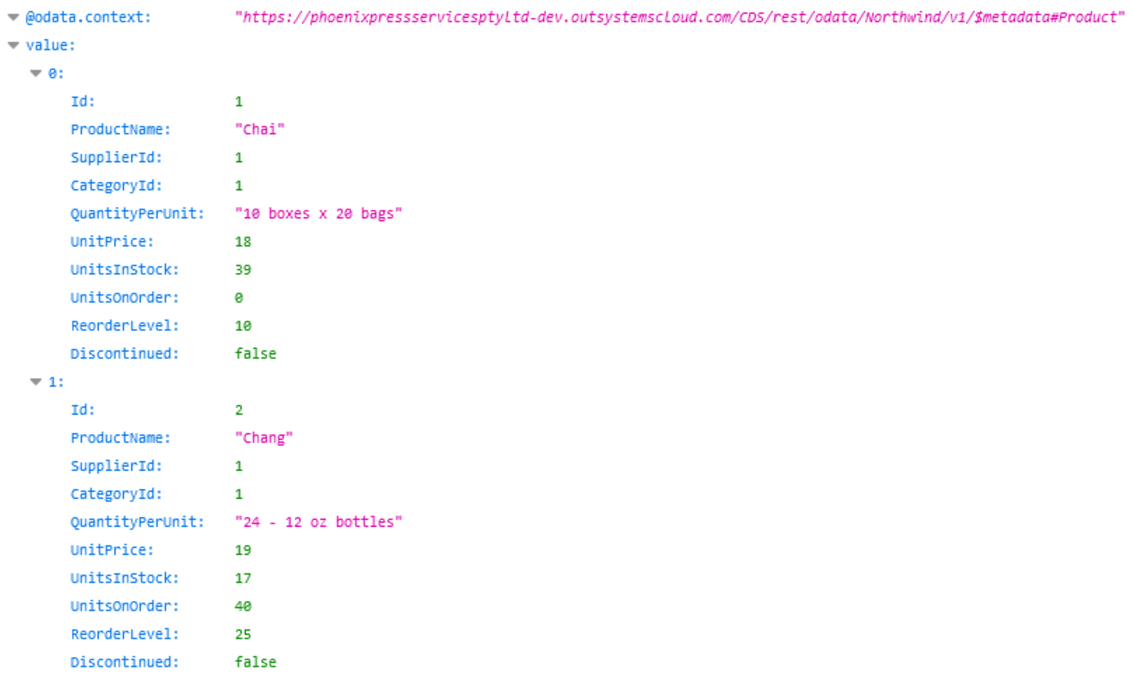
Table 5.3: Mapping query options into SQL statements, through a query model.

protocol. The inline *count* option is mapped into a boolean value and a second SQL statements is run to extract the count value. Each filter condition is added to the where conditions list in our model. The selected columns are also added to the designated list. The *search* query option gets modelled into as much filter conditions using the like operator as the number of columns in the target resource. Finally, an expansion is similar to requesting a relationship between to entities, but instead of only exposing the end entity, the end entity is exposed within the source entity. This is done by adding the end entity to the join tables list.

5.2.3 De/Serialization Process

This process creates an OData serializer and deserializer that supports both the OData JSON and XML formats.

In order to construct the textual representation of the OData records according to the protocol's



```
▾ @odata.context:      "https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/v1/$metadata#Product"
▾ value:
  ▾ 0:
        Id:                1
        ProductName:       "Chai"
        SupplierId:        1
        CategoryId:        1
        QuantityPerUnit:   "10 boxes x 20 bags"
        UnitPrice:         18
        UnitsInStock:      39
        UnitsOnOrder:      0
        ReorderLevel:      10
        Discontinued:      false
  ▾ 1:
        Id:                2
        ProductName:       "Chang"
        SupplierId:        1
        CategoryId:        1
        QuantityPerUnit:   "24 - 12 oz bottles"
        UnitPrice:         19
        UnitsInStock:      17
        UnitsOnOrder:      40
        ReorderLevel:      25
        Discontinued:      false
```

Figure 5.2: Example of a Product collection.

norms, the serializer applies a model-to-text transformation to the OData query result [14]. For example, an entity is represented by a JSON object representing its properties, composed of list of key/value pairs; and an entity collection is transformed to a JSON array holding the entities. The serializer additionally takes into consideration the query model while generating the JSON representation, for example the number of key/value pairs correspond to the number of selected columns. Fig. 5.2 depicts an example of a Product collection. Apart from the entity's properties, the JSON object also includes the annotation *odata.context* as metadata, which provides the payload's root context URL.

The deserializer parses and processes the body of OData requests POST and PUT to construct the details of the INSERT and UPDATE SQL queries [14]. In the resulting SQL statement, each key/value pair in a JSON object is transformed to the corresponding field in the relevant database and its respective value. For DELETE requests (see Table 3.3), we only need the resource path component in the URL that targets a specific entity within a collection to be removed.

The XML representation format follows a similar procedure for the metadata document of the OData service.

6

Demonstration

Contents

6.1 Context 45

6.2 Exposing an OData service in OutSystems 45

6.3 OutSystems CDS Project 49

This chapter is related to the DSRM demonstration phase and illustrates how our research proposal is used to solve the research problem described in Chapter 4. To demonstrate that the proposal can be used to solve the research problem, we developed this method in a specific context, using a particular Low-Code Development Platform and Business Intelligence tool to integrate them through a dynamic approach exposing an OData Service.

6.1 Context

This master's thesis was implemented in a professional environment, integrated in a company dedicated to delivering digital solutions through developing cutting-edge enterprise software, *PhoenixDX* [34].

The Low-Code Development Platform used for this validation was OutSystems [4]. OutSystems is a modern low-code application platform that speeds up the creation of applications while also providing exceptional flexibility and efficiency [4]. It enables the development of desktop and mobile applications which may run either in the cloud or on local infrastructures. OutSystems has three significant components:

- Integration Studio: allows database connections through either Java or .NET
- Service Studio: where the behaviour of the application being developed is specified
- Platform Server: the cloud server used to develop, orchestrating all runtime, deployment, and managing activities for all applications

In order to validate our OData service, we consumed it through PowerBI [35]. PowerBI allows to connect and visualize any data using the unified, scalable platform for self-service and enterprise Business Intelligence tool [35]. It is a tool that is easy to use and helps gaining a deeper data insight.

To the extent of our knowledge, OutSystems has no built-in component to support exposing OData services.

6.2 Exposing an OData service in OutSystems

According to our proposed approach in Chapter 5, the integration process involves three steps (check Table 5.1), which are described in the following sections.

6.2.1 Creation of OData's EDM

To expose an OData service dynamically, we used OutSystems' metamodel. The metamodel specifies what can be found in the model, from the data migration point of view. An OutSystems application is made up of modules defined in Service Studio. Modules allow you to structure your application into several pieces, implementing a specific purpose per piece. Every entity created and used in an application is related to the module where it is defined.

When creating an entity within a module, you are to give it a logical name. Given that name, the platform automatically creates internally a physical name. The mapping between the physical table name and the logical name is found in a system entity in the metamodel and each entity is related to its defining module. Besides this table, there is also another system entity that contains all attributes that can be identified based on the entity id they are related to. With this information, we are capable of identifying every entity and its own attributes defined in a specific module. Were a module to create a new table, any application would be capable of dynamically extracting information about the new table, using the system entities in the metamodel.

To expose an OData service, the OutSystems application must expose a REST service. Such application exposes a REST service with 3 main procedures exposing the service document, the metadata document, and result of any querying or modifying the data. The OData service endpoint of the application has a defined context, which contains the required configuration for the service to operate. The service context contains:

- ServiceRoot - the absolute URL of the Service Document
- EntityDefinitions - the definition of all entities and their attributes, exposed by the service
- Body - the HTTP request body for POST and PUT requests

Each OData request, in order to execute, needs access to the service context and to its inputs. The input for a given OData request is the path of the request, which is a possibly empty string starting after the service root and also including any query string of the request, and the body to insert or update data. The path is composed of the target resource path and query options. When the path is empty the service document is returned, and the metadata document is returned when the path is *$metadata*.

Microsoft's OData Core library [36] is designed to read and write all kinds of OData payloads, such as service document, model metadata, entity set and references, etc. Through a .NET extension of our OutSystems application we are able create and read such payloads. To build the metadata XML, an EdmModel must first be built, through the OData Core library. In the OData library, the object that represents all of the entities exposed on an OData service is called EdmModel. This contains a list of entities, in a similar fashion to the list of entity definitions. Given the EdmModel and the service root, writing metadata is simple by using *ODataMessageWriter.WriteMetadataDocument()*. The service document can also be generated automatically. The Service Document and the Metadata Document, are requests simpler to execute since they only depend on the EdmModel and service root. Writing other payloads is a more complex process, which is further detailed in the following sections.

6.2.2 Mapping between OData requests and SQL statements

In order to execute a dynamic OData request we need to parse the path information. For this task, we used the ODataUriParser class provided by the OData library. The ODataUriParser provides detailed information of the multiple segments of the given Path, for example which entity type it refers to, and if it has a key predicate. It also parses expressions related to the query options keywords such as $filter, $orderby, $top, $skip, etc. However, this information is not directly suitable to generate an SQL statement. To transform the output of ODataUriParser into SQL, we use a QueryParser class. This class first builds a QueryModel object, which contains information directly suitable to generate the SQL statement. This object contains information such as the list of selected columns, the list of tables in the FROM clause, the list of conditions on the WHERE clause, etc. Once obtained the QueryModel, transforming it into SQL is a simple operation of mostly concatenating strings. We defined a specific query model to help the mapping to SQL statements for each data transformation operation (i.e., create, read, update and delete).

The query model defined in Section 5.2.2 is used for the GET requests.

Listing 6.1: Example of an OData GET request mapping to SQL statement.

```
HTTP Method: GET
URL: https://myhost/Northwind.svc/Product(2)?$filter=Price+gt+15&$orderby=Name
Query Model:
  List of output tables  →  {Product}
  List of where conditions  →  {(Price > 15)}
  List of orderby conditions  →  {(Name, ASC)}
SQL:
  SELECT *
  FROM Product
  WHERE Product.Price > 15
  ORDER BY Product.Name
```

For DELETE requests the query model contains an entity type and a key/value pair. The entity type is the entity table from which the record is to be deleted, and the key/value pair defines the primary key and the corresponding value to specify the record. These values are retrieved from the target resource segment of the URL path. Listing 6.2 illustrates an example of a DELETE request.

Listing 6.2: Example of an OData DELETE request mapping to SQL statement.

```
HTTP Method: DELETE
URL: https://myhost/Northwind.svc/Product(2)
Delete Model:
  Entity Type  →  Product
  Key/Value Pair  →  (Id,2)
SQL:
  DELETE FROM Product
  WHERE Product.Id = 2
```

To insert a new record through the OData service, the query model is composed of an entity type and a list of key/value pairs. Again, the entity type is the entity table from which the new record is to be inserted. The list of key/value pairs have the name and value of all the attributes of that specific entity type. The entity type and the key/value pairs are specified in the HTTP request's body. Listing 6.3 illustrates an example of a POST request.

Listing 6.3: Example of an OData POST request mapping to SQL statement.

```
HTTP Method: POST
URL: https://myhost/Northwind.svc/Product
Body:
  {
  "@odata.type": "Northwind.Product",
  "Name": "Chai",
  "Price": "20.00",
  "CategoryId": "1"
  }
Insert Model:
  Entity Type  →  Product
  List Key/Value Pairs  →  {(Name,'Chai'), (Price, 20.00), (CategoryId, 1)}
SQL:
  INSERT INTO Product(Product.Name, Product.Price, Product.CategoryId)
  VALUES ('Chai', 20.00, 1)
```

For PUT requests the query model is the similar to the insert model. However the key/value pairs are a subset of the entity type's attributes and corresponding values. In addition, there is a specific key/value pair in the model that refers to a specific record, through the primary key. Once more, the entity type and the key/value pairs are specified in the HTTP request's body. Listing 6.4 illustrates an example of a PUT request.

Listing 6.4: Example of an OData PUT request mapping to SQL statement.

```
HTTP Method: PUT
URL: https://myhost/Northwind.svc/Product(2)
Body:
  {
  "@odata.type": "Northwind.Product",
  "Price": "23.99"
  }
Update Model:
  Entity Type  →  Product
  Key/Value Pair  →  (Id, 2)
  List Key/Value Pairs (Properties)  →  {(Price, 23.99)}
SQL:
  UPDATE Product
  SET Product.Price = 23.99
  WHERE Product.Price = 2
```

After obtaining the SQL statement, we execute it using OutSystems' database API. Such API allows to retrieve an object of type IDataReader after executing the SQL command. This object, which defines the query result, can then be used to read multiple rows, and multiple columns on each row.

6.2.3 De/Serialization Process

The serialization of the XML metadata document and the JSON service document were already described in the end of Section 6.2.1. For other types of OData payloads, we defined a specific output class to help the the serialization process for each data transformation operation (i.e., GET, POST, PUT, DELETE).

For updating and removing records, the payload has no content, therefore the serialization process only requires setting the status code of the response to 204 (indicating no content).

When querying the data exposed by the service, the output class receives the query model completed in the previous section and the data table with the resulting records of such query. To serialize into an OData payload, we used the ODataMessageWriter class provided by the OData Core library. The ODataMessageWriter is a class for writing OData payloads. A collection of entities is described by the ODataResourceSet class and for each data row of the query result we define a list of ODataProperty class to describe the entity's properties. In case there are nested entities that were expanded using the $expand query option, an ODataResource is used to describe them. The status code is set to 200 (OK) upon a successful request.

For new data records being inserted, the output class only receives the query model completed in the previous section. From the model, the properties and respective values of the record are extracted and converted into ODataProperty and written through the ODataMessageWriter. The status code is set to 201 (created).

The deserializer parses and processes the body of OData requests to insert and update data, to build the details of query model. For this, we use the ODataMessageReader class which is provided by the OData Core library. The ODataMessageReader is a class for reading all OData payloads. It is created using the request's body and the body is parsed using an ODataReader to read a resource within the body and proceed to filling in the corresponding query model.

Listing 6.5 represents the execution of OData GET, DELETE, POST and PUT requests in pseudo-code.

Listing 6.5: Pseudo-code of OData requests execution.

```
ExecuteGET(model, serviceRoot, path):
  if (path == ""):
    return ServiceDocument(model, serviceRoot)
  else if (path == "\$metadata"):
    return MetadataDocument(model, serviceRoot)
  else:
    return Output(Query(Parse(model, path)), serviceRoot)

ExecuteDELETE(model, serviceRoot, path):
  return Output(Delete(Parse(model, path)), serviceRoot)

ExecutePOST(model, serviceRoot, path, body):
  return Output(Insert(Parse(model, path, body)), serviceRoot)

ExecutePUT(model, serviceRoot, path):
  return Output(Update(Parse(model, path, body)), serviceRoot)
```

6.3 OutSystems CDS Project

Once we were able to expose an OData service that can query and manipulate data using CRUD operations, a generic application was developed called Common Data Service (CDS) Project. CDS

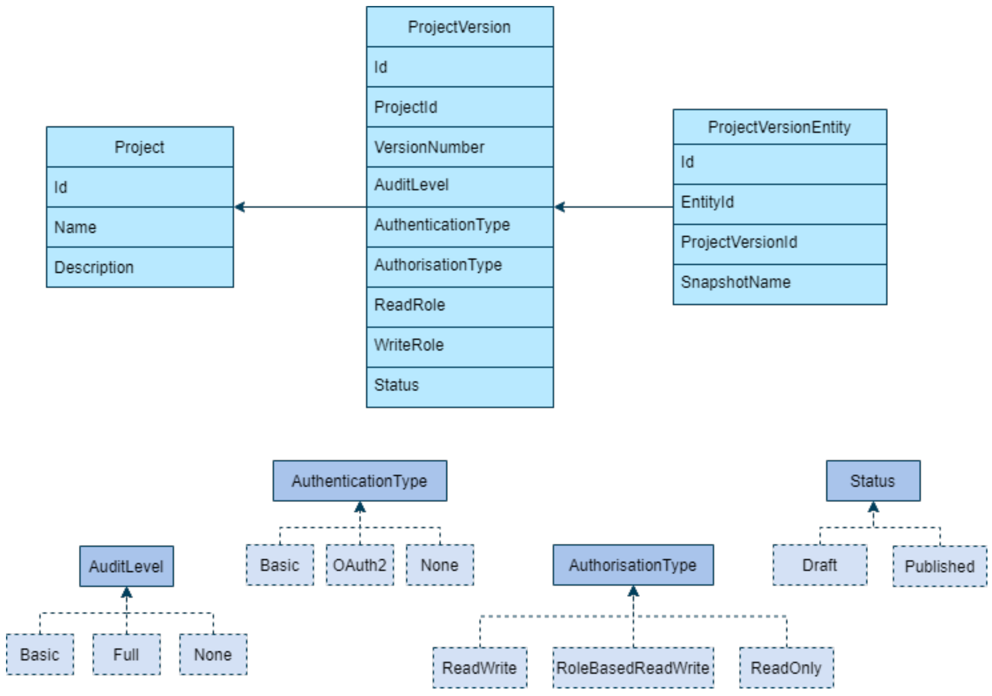


Figure 6.1: Entity diagram of CDS Project.

Project allows authenticated users to create projects with whichever entities from the current OutSystems environment. A project has a name and a description and has one or more project versions associated to it (check Fig. 6.1). Each project version is related to a single project, has a version number and multiple definitions regarding to auditing and security. A project version status is either draft or published.

In project versions with no authentication, the endpoint does not enforce any authentication. The endpoint enforces HTTP Basic authentication, when AuthenticationType is basic. Project versions also allow OAuth 2.0 authentication which is the most secure authentication type of the three, based on short-lived access tokens.

ReadOnly AuthorisationType allows authorised users to access entities for reading information, but not writing. ReadWrite allows user to access entities for reading and writing information. RoleBasedReadWrite allows authorised users based on applicational roles to read and write information. Such roles are specified in the project version's ReadRole and WriteRole. Finally, the user is able to specify an Audit Level. When a user selects a Basic AuditLevel, the user's details and the request payload are captured for all requests. For a Full AuditLevel, user details, the request and response payloads are captured for all requests.

For each entity the project version exposes, a ProjectVersionEntity is created. When a version is published and one of the exposed entities has been modified or deleted, the live version still keeps the unchanged entity through the SnapshotName attribute.

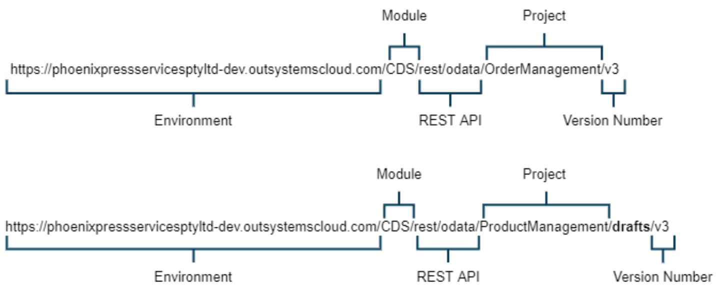
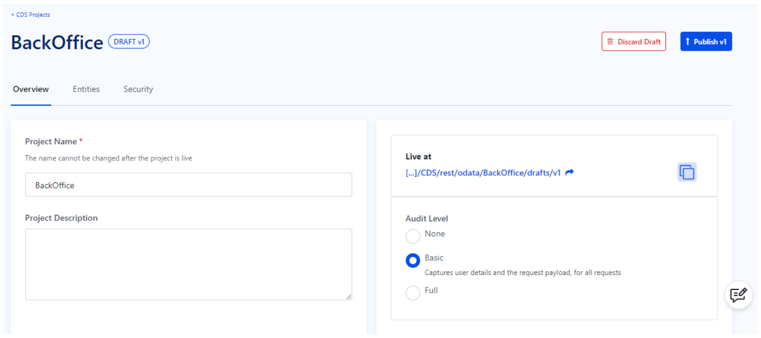

Module
Project
https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/OrderManagement/v3
Environment
REST API
Version Number
Module
Project
https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/ProductManagement/drafts/v3
Environment
REST API
Version Number

Figure 6.2: URL components of published and draft projects.



Project Name *
The name cannot be changed after the project is live
BackOffice

Project Description

Live at
[...]/CDS/rest/odata/BackOffice/drafts/v1

Audit Level
None
[✓ Basic]
Captures user details and the request payload, for all requests
Full

Figure 6.3: UI of the Overview screen when creating a new project.

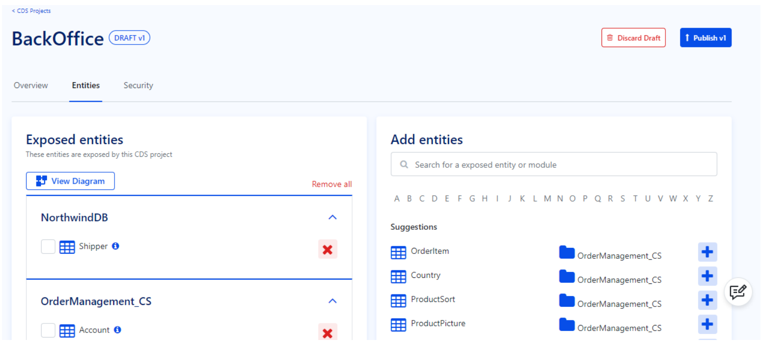
Each project version (draft or not) has a live OData service exposing the related entities. Any authorised user is able to access it via a given URL. The OData service is exposed through the OutSystems CDS application, which given a project version exposes all the related entities. The endpoint contains the name of the project and the version number, as illustrated in Fig. 6.2. For draft versions, the keyword *drafts* is included in the URL.

6.3.1 Example

When users create new projects, the screen they see is the Overview, shown in Fig. 6.3. Here the user defines the name of the project as it is a new one (for new versions of existing projects the name cannot be changed) and the description of the version. The OData service endpoint exposing the project's entities is live at the indicated URL. Users can also select the AuditLevel desired for the current project version.

To select entities, the user must change to the Entities screen, shown in Fig. 6.4. Users are allowed to add and remove entities, from all the active modules of the current OutSystems environment. Some entities are suggested based on the projects exposed entities. In our example, we can check that entities from at least two different modules (NorthwindDB and OrderManagement_CS) are being exposed.

Finally, on the last screen Security (Fig. 6.5) users are able to select the authentication and


< CDS Projects

BackOffice DRAFT v1

Discard Draft | Publish v1

Overview | Entities | Security

Exposed entities

These entities are exposed by this CDS project

View Diagram | Remove all

NorthwindDB
- Shipper

OrderManagement_CS
- Account

Add entities

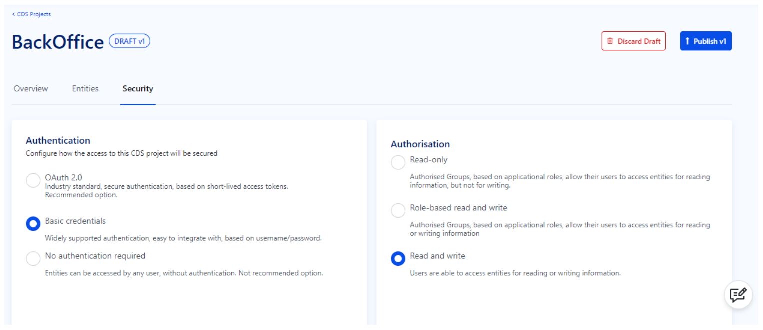
Search for a exposed entity or module

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Suggestions

OrderItem	OrderManagement_CS
Country	OrderManagement_CS
ProductSort	OrderManagement_CS
ProductPicture	OrderManagement_CS

Figure 6.4: UI of the Entities screen when creating a new project.


< CDS Projects

BackOffice DRAFT v1

Discard Draft | Publish v1

Overview | Entities | Security

Authentication

Configure how the access to this CDS project will be secured

- OAuth 2.0 — Industry standard, secure authentication, based on short-lived access tokens. Recommended option.
- [✓ Basic credentials] — Widely supported authentication, easy to integrate with, based on username/password.
- No authentication required — Entities can be accessed by any user, without authentication. Not recommended option.

Authorisation

- Read-only — Authorised Groups, based on applicational roles, allow their users to access entities for reading information, but not for writing.
- Role-based read and write — Authorised Groups, based on applicational roles, allow their users to access entities for reading or writing information
- [✓ Read and write] — Users are able to access entities for reading or writing information.

Figure 6.5: UI of the Security screen when creating a new project.

authorisation types. At any given moment, the user might discard the current draft or publish it, making it a live project version.

7 Evaluation

Contents

7.1 Test Application 55
7.2 Unit Tests 56

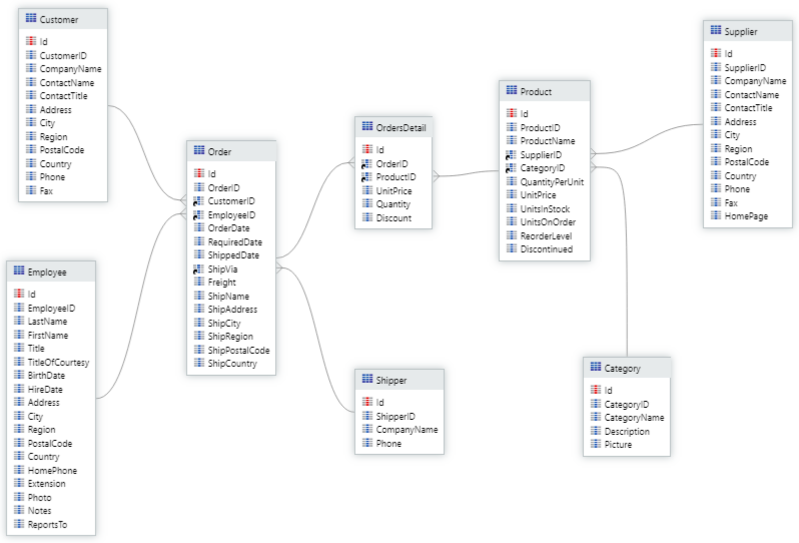


Figure 7.1: The Northwind DB entity diagram

This Chapter addresses the evaluation phase of DSRM. Demonstrating the artifact's use in one or more cases is a standard means of ensuring that the artifact meets its goal [37]. For evaluating the efficacy of our artifact, we demonstrate its response in several use cases [37]. As we were developing our OutSystems solution, Unit tests were incrementally added to assess our artifact in specific scenarios.

7.1 Test Application

In all of the following use cases, the test application we used was a project exposing the Northwind Database (DB). The Northwind Database is a sample database that was created to exhibit the performance of Microsoft's products. The database contains sales information of a hypothetical company that exports/imports specialty foods called Northwind. The DB being assessed has a simple data model that is illustrated in Fig. 7.1.

A CDS project was created named Northwind, with all Northwind DB's entities. No authentication is required to access the service and it is live in `https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2`.

7.2 Unit Tests

7.2.1 Service and Metadata Documents

Being able to expose the service and metadata documents is a requirement for any OData service. The service document illustrated in Listing B.1 is retrieved with a GET request on the service root URI. This JSON document lists the collections of available resources provided by the project's service. Listing C.1 is the response of a GET request to the metadata document, by appending the segment *$metadata* to the service root URI. It describes each entity type of the project, listing their properties and corresponding data types.

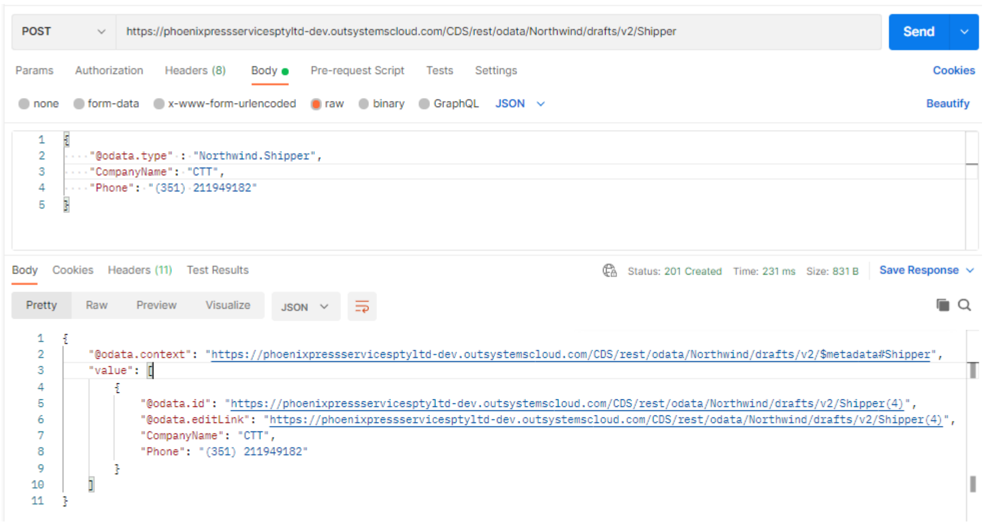
7.2.2 CRUD Operations

One of our goals was to be able to manipulate the data exposed in the service using CRUD operations. First we decided to read the list of available shippers using a GET request on `https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Shipper`. The response of the request is listed in Listing 7.1. The project has three shipping companies.

Listing 7.1: GET request on Shipper entity set.

```
{
    "@odata.context":"https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/
        rest/odata/Northwind/drafts/v2/$metadata#Shipper",
    "value":
        [
            {
                "Id":1,
                "CompanyName":"Speedy Express",
                "Phone":"(503) 555-9831"
            }
            {
                "Id":2,
                "CompanyName":"United Package",
                "Phone":"(503) 555-3199"
            }
            {
                "Id":3,
                "CompanyName":"Federal Shipping",
                "Phone":"(503) 555-9931"
            }
        ]
}
```

For a shipping company to be added to the Shipper collection, the service client must send a POST request to that collection's URL (`https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Shipper`). The POST body has to consist of a single valid entity representation. To test this, we used Postman. Postman is an API platform for both building and using APIs [38]. It allows you to send HTTP requests to web APIs, including simulating requests with bodies which is important for POST and PUT request methods. The request (and corresponding response) in Fig 7.2 creates a Shipper entity whose company name is *CTT*. To check that



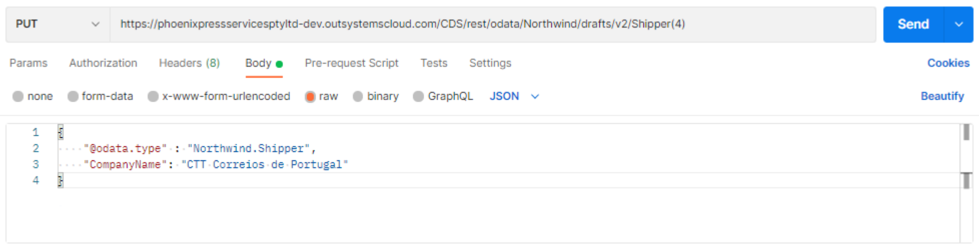
```
https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Shipper
```

```
{
    "@odata.type" : "Northwind.Shipper",
    "CompanyName": "CTT",
    "Phone": "(351) 211949182"
}
```

Status: 201 Created   Time: 231 ms   Size: 831 B

```
{
    "@odata.context": "https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/$metadata#Shipper",
    "value": [
        {
            "@odata.id": "https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Shipper(4)",
            "@odata.editLink": "https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Shipper(4)",
            "CompanyName": "CTT",
            "Phone": "(351) 211949182"
        }
    ]
}
```

Figure 7.2: POST request and response of creating a new entity in Shipper collection, using Postman.



```
https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Shipper(4)
```

```
{
    "@odata.type" : "Northwind.Shipper",
    "CompanyName": "CTT Correios de Portugal"
}
```

Figure 7.3: PUT request for updating a specific entity in Shipper collection, using Postman.

the entity has in fact been created, we typed the *odata.id* link and got the JSON listed in Listing 7.2.

Listing 7.2: Shipper single entity with id=4.

```
{
    "@odata.context":"https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/
        rest/odata/Northwind/drafts/v2/$metadata#Shipper",
    "value":
        [
            {
                "Id":4,
                "CompanyName":"CTT",
                "Phone":"(351) 211949182"
            }
        ]
}
```

To update an entity within a collection, the service client must send a PUT request to the specific entity's URL (`https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Shipper(4)`). The PUT body has to consist of a single valid entity representation, composed of the properties the client desires to update. To test this functionality, we a PUT request using Postman, represented in Fig. 7.3. PUT requests for OData services have no content responses. Once more, to check our data has successfully been updated, we requested the specific Shipper entity with id=4 and got the JSON listed in Listing 7.3.

Listing 7.3: Shipper single entity with id=4, after updating the company's name.

```
{
    "@odata.context":"https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/
        rest/odata/Northwind/drafts/v2/$metadata#Shipper",
    "value":
        [
            {
                "Id":4,
                "CompanyName":"CTT Correios de Portugal",
                "Phone":"(351) 211949182"
            }
        ]
}
```

To remove an entity within a collection, one must send a DELETE request with the primary key of the entity to be removed. The request bellow deletes the Shipper with id = 4.

DELETE `https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Shipper(4)`

DELETE requests for OData services have no content responses. Again, to check our data has successfully been removed, we requested the specific Shipper entity with id=4 and got an empty value JSON.

7.2.3 Querying Requests

All of the query options in Table 5.3 were tested for simple cases and composed two by two. The $filter query option was tested for different data types and data comparisons as well. We will instance a small portion of the unit tests over querying the data, as more than 100 tests were developed.

There are a total of 77 product instances in the previously created Northwind project. As default the products are ordered by id number. In order to know the top 3 most expensive beverages we created a GET request with the following URL: `https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Product?$filter=CategoryId+eq+1&$orderby=UnitPrice+desc&$top=3`. Out of the 8 defined categories, beverages entity has id=1 and to get the most expensive ones we ordered the results by UnitPrice descending, selecting only the top 3. To simplify the result, we only select three properties: the name of the product, the price per unit and the quantity per unit (by adding $select=ProductName,UnitPrice,QuantityPerUnit to the query options). The result of this query with only three selected columns is in Listing 7.4.

Listing 7.4: Two of the top 3 most expensive beverages.

```
{
    "@odata.context":"https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/
        rest/odata/Northwind/drafts/v2/$metadata#Product",
    "value":
        [
            {
                "ProductName":"C\u00f4te de Blaye",
                "UnitPrice":263.50000000,
                "QuantityPerUnit":"12 - 75 cl bottles"
            }
            {
                "ProductName":"Ipoh Coffee",
                "UnitPrice":46.00000000,
                "QuantityPerUnit":"16 - 500 g tins"
            }
            {
                "ProductName":"Chang",
                "UnitPrice":19.00000000,
                "QuantityPerUnit":"24 - 12 oz bottles"
            }
        ]
}
```

Navigating through relationships was also tested. Every OrderDetail entity has a corresponding product, which has a category associated with it. In order to get what is the category of the product from the order detail with id=2, the following request was made: `https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/OrderDetail(2)/Product/Category`. In this use case, two navigation properties were tested, from OrderDetail to Product and from Product to Category. The response of this request is listed in Listing 7.6.

Listing 7.5: Checking the category of the product belonging to order detail with id=2.

```
{
    "@odata.context":"https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/
        rest/odata/Northwind/drafts/v2/$metadata#Category",
    "value":
        [
            {
                "Id":5,
                "CategoryName":"Grains/Cereals",
                "Description":"Breads, crackers, pasta, and cereal",
                "Picture":"FRwvAAIAAAANAA4AFAAhAP////9CaXRtYXAgSW1hZ2UAUGFpbnQuUGlj
                    ...CICwAAiAsAAAgAAA=="
            }
        ]
}
```

To test expanding entities within entities, we decided to check products with Chef names and respective categories that haven't been discontinued, through `https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Product?$search=Chef&$expand=Category&$filter=Discontinued+eq+false`. To search products with Chef names on it we used $search=Chef, expanded the category within the products with the $expand query option and filtered the results for Discontinued=false. The result of this request is listed in Listing 7.6.

Listing 7.6: Products with Chef names that aren't discontinued, with the respective categories.

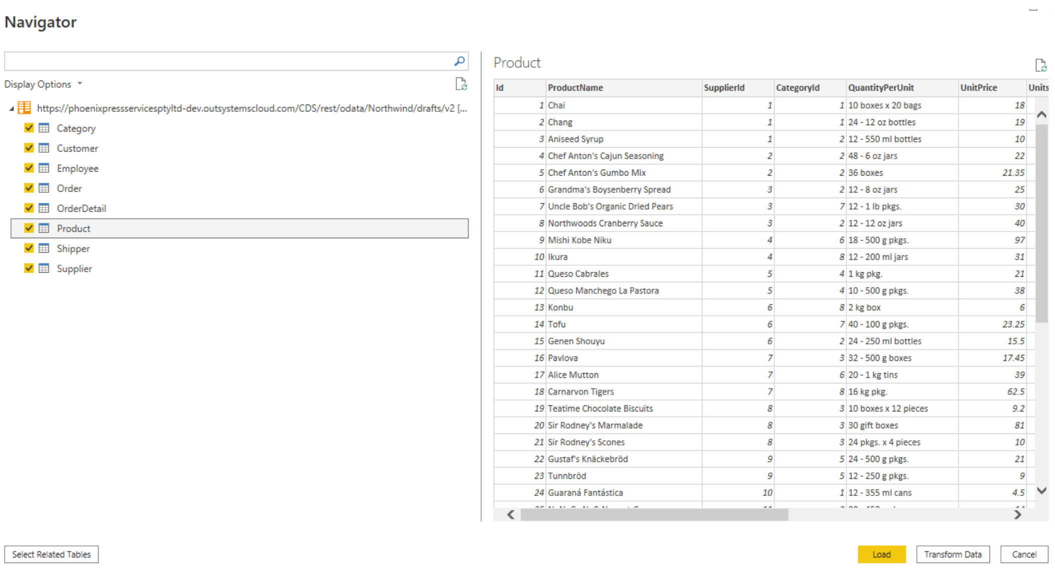
```
{
    "@odata.context":"https://phoenixpressservicesptyltd-dev.outsystemscloud.com
        /CDS/rest/odata/Northwind/drafts/v2/$metadata#Product",
    "value":
        [
            {
                "Id":4,
                "ProductName":"Chef Anton's Cajun Seasoning",
                "SupplierId":2,
                "CategoryId":2,
                "QuantityPerUnit":"48 - 6 oz jars",
                "UnitPrice":22.00000000,
                "UnitsInStock":53,
                "UnitsOnOrder":0,
                "ReorderLevel":0,
                "Discontinued":false,
                "Category":
                    {
                        "Id":2,
                        "CategoryName":"Condiments",
                        "Description":"Sweet and savory sauces, relishes,
                            spreads, and seasonings",
                        "Picture":"FRwvAAIAAAANAA4AFAAhAP////9
                            CaXRtYXAgSW1hZ2UAUGFpbn...AACICwAAiAsAAAgAAA=="
                    }
            }
        ]
}
```

The URL `https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Product?$filter=UnitsInStock+eq+0+and+UnitsOnOrder+gt+0` queried which products that were out of stock and awaiting order arrivals, by filtering both UnitsInStock and UnitsOnOrder properties. The result was a single product, described in Listing 7.7.

Listing 7.7: Product out of stock awaiting order arrivals.

```
{
    "@odata.context":"https://phoenixpressservicesptyltd-dev.outsystemscloud.com
        /CDS/rest/odata/Northwind/drafts/v2/$metadata#Product",
    "value":
        [
            {
                "Id":31,
                "ProductName":"Gorgonzola Telino",
                "SupplierId":14,
                "CategoryId":4,
                "QuantityPerUnit":"12 - 100 g pkgs",
                "UnitPrice":12.50000000,
                "UnitsInStock":0,
                "UnitsOnOrder":70,
                "ReorderLevel":20,
                "Discontinued":false
            }
        ]
}
```

Moreover, to check both the count of the total number of products and the second cheapest product, we requested the following URL: `https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/Product?$orderby=UnitPrice&$top=1&$skip=1&$count=true`. The inline count is done through $count. Note that the inline count is not affected by $top and $skip query options. We also ordered the result by unit price and the default sort order is ascending. To get the second cheapest, after ordering the results we requested the top 1 with one record to skip. The request response is listed in Listing 7.8.



Id	ProductName	SupplierId	CategoryId	QuantityPerUnit	UnitPrice	Units
1	Chai	1	1	10 boxes x 20 bags	18	
2	Chang	1	1	24 - 12 oz bottles	19	
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10	
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22	
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35	
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25	
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30	
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40	
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97	
10	Ikura	4	8	12 - 200 ml jars	31	
11	Queso Cabrales	5	4	1 kg pkg.	21	
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38	
13	Konbu	6	8	2 kg box	6	
14	Tofu	6	7	40 - 100 g pkgs.	23.25	
15	Genen Shouyu	6	2	24 - 250 ml bottles	15.5	
16	Pavlova	7	3	32 - 500 g boxes	17.45	
17	Alice Mutton	7	6	20 - 1 kg tins	39	
18	Carnarvon Tigers	7	8	16 kg pkg.	62.5	
19	Teatime Chocolate Biscuits	8	3	10 boxes x 12 pieces	9.2	
20	Sir Rodney's Marmalade	8	3	30 gift boxes	81	
21	Sir Rodney's Scones	8	3	24 pkgs. x 4 pieces	10	
22	Gustaf's Knäckebröd	9	5	24 - 500 g pkgs.	21	
23	Tunnbröd	9	5	12 - 250 g pkgs.	9	
24	Guaraná Fantástica	10	1	12 - 355 ml cans	4.5	

Figure 7.4: Getting OData feed data in a PowerBI report.

Listing 7.8: Inline count and second cheapest product query result.

```
{
    "@odata.context":"https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/rest/odata/Northwind/drafts/v2/$metadata#Product",
    "@odata.count":77,
    "value":
        [
            {
                "Id":24,
                "ProductName":"Guaran\u00e1 Fant\u00e1stica",
                "SupplierId":10,
                "CategoryId":1,
                "QuantityPerUnit":"12 - 355 ml cans",
                "UnitPrice":4.50000000,
                "UnitsInStock":20,
                "UnitsOnOrder":0,
                "ReorderLevel":0,
                "Discontinued":true
            }
        ]
}
```

7.2.4 PowerBI Integration

Finally, to further validate our project's OData service and the reason why it was automatically generated, we consumed it through PowerBI. In order to complete this integration, when a PowerBI report is created the data source to get the data from has to be an OData feed. By inserting the project's service root URL we are able to load all the project's data into the report. After selecting the necessary authentication requirements, we are able to select all or a subset of the tables listed in the OData feed, as shown in Fig. 7.4. To ensure the data was successfully loaded we created a simple report listing the number of orders per customer's country, shown in Fig. 7.5. By analyzing the report, USA had the most amount of customer orders, followed by Germany and Brazil.

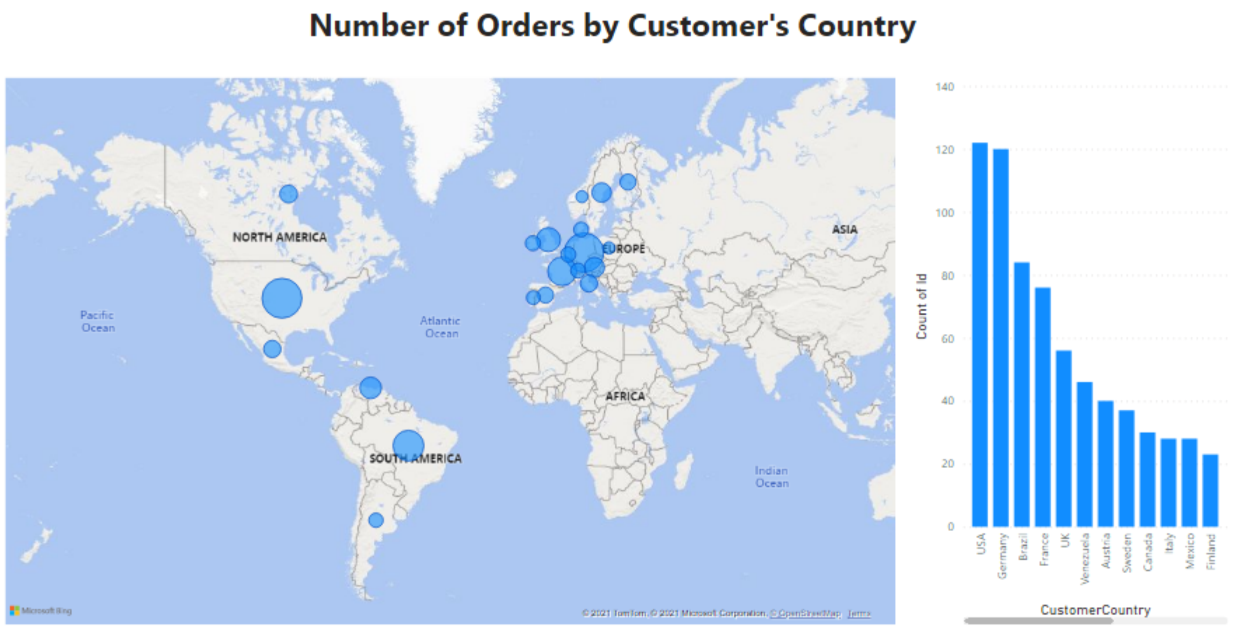


Figure 7.5: PowerBI report with the project's OData feed loaded data.

8 Conclusion

Contents

8.1 Research Contributions 65

8.2 Research Limitations 65

8.3 Future Work 65

8.1 Research Contributions

Overall, LCDPs are useful for organizations that have limited budget and IT resources since fully-featured products can be delivered in a short amount of time. Third-party integration of the developed applications can be hampered depending on the extensibility capabilities of the employed LCDPs.

With the growing relevance of low-code platform applications in enterprise landscapes it is fundamental that its applicational data is made available and interoperable in the speed of low-code. This is a significant gap and as far we are concerned none of the most used LCDPs have a solution in their short to medium term road-map. Such integration can be made possible with an OData service.

We applied the Design Science Research Methodology to develop an artifact that would solve our research problem, stated in Chapter 4. Such artifact creates an API exposing the data retrieved from the LCDP application as an OData service. Not only does this allow end-users to easily get the information in need through the OData query language, but it also enables the data to be consumed by other applications. Our proposal was demonstrated using OutSystems applications and the PowerBI tool as the systems being integrated through OData services. To evaluate the efficacy of our artifact, we demonstrate its response in several use cases, checking that the stated objectives were accomplished.

Low-code applications are now interoperable with the outside world through the OData protocol. Not only can the data be updated without having access to the app but the data can be further consumed by other tools and frameworks.

8.2 Research Limitations

Although the OData service is created automatically with our artifact, we still had to add a security layer in our demonstration, since OData by itself is not secured in a way that feels acceptable, especially being an open data initiative. Furthermore, we haven't tested the systems integration with large datasets.

8.3 Future Work

As future work, to facilitate the design and creation of more sophisticated aspects, we want to extend our mapping features to capture further OData behavioral elements such as functions and actions. We intend to expand our approach to include all features of the advanced OData conformance level.

Bibliography

[1] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[2] A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio, “Supporting the understanding and comparison of low-code development platforms,” in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2020, pp. 171–178.

[3] “What is low-code? a full guide to low-code platforms | creatio,” https://www.creatio.com/page/low-code, (Accessed on 09/2021).

[4] “Build applications fast, right and for the future | outsystems,” https://www.outsystems.com/, (Accessed on 09/2021).

[5] “Low-code application development platform - build apps fast & efficiently | mendix,” https://www.mendix.com/, (Accessed on 09/2021).

[6] “Appian: Low-code automation | business apps | bpm | rpa,” https://appian.com/, (Accessed on 09/2021).

[7] “Kissflow - a unified digital workplace | all in one platform,” https://kissflow.com/, (Accessed on 09/2021).

[8] R. Waszkowski, “Low-code platform for automating business processes in manufacturing,” *IFAC-PapersOnLine*, vol. 52, no. 10, pp. 376–381, 2019.

[9] “What is paas? platform as a service | microsoft azure,” https://azure.microsoft.com/en-us/overview/what-is-paas/, (Accessed on 09/2021).

[10] S. Keele *et al.*, “Guidelines for performing systematic literature reviews in software engineering,” Citeseer, Tech. Rep., 2007.

[11] M. J. Carey, N. Onose, and M. Petropoulos, “Data services,” *Communications of the ACM*, vol. 55, no. 6, pp. 86–97, 2012.

[12] G. Harrison, “Data marketplaces have yet to deliver on early promise - database trends and applications,” https://www.dbta.com/Columns/Big-Data-Notes/Data-Marketplaces-Have-Yet-to-Deliver-on-Early-Promise-98512.aspx, 2014, (Accessed on 06/2021).

[13] S. Burgess, “Open data protocol - build great experiences on any device with odata | microsoft docs,” https://docs.microsoft.com/en-us/archive/msdn-magazine/2011/september/open-data-protocol-build-great-experiences-on-any-device-with-odata, 2011, (Accessed on 06/2021).

[14] H. Ed-Douibi, J. L. C. Izquierdo, and J. Cabot, “Model-driven development of odata services: An application to relational databases,” in *2018 12th International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 2018, pp. 1–12.

[15] C.-Y. Huang and S. Liang, “A sensor data mediator bridging the ogc sensor observation service (sos) and the oasis open data protocol (odata),” *Annals of GIS*, vol. 20, no. 4, pp. 279–293, 2014.

[16] H. Ed-Douibi, J. L. C. Izquierdo, and J. Cabot, “Apicomposer: Data-driven composition of rest apis,” in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2018, pp. 161–169.

[17] M. Thoma, T. Kakantousis, and T. Braun, “Rest-based sensor networks with odata,” in *2014 11th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*. IEEE, 2014, pp. 33–40.

[18] “Entity data model - ado.net | microsoft docs,” https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/entity-data-model, (Accessed on 09/2021).

[19] S. Burgess, “Practical odata - building rich internet apps with the open data protocol | microsoft docs,” https://docs.microsoft.com/en-us/archive/msdn-magazine/2010/june/practical-odata-building-rich-internet-apps-with-the-open-data-protocol, 2010, (Accessed on 06/2021).

[20] “Wso2 joins technology leaders in proposing oasis odata technical committee,” https://wso2.com/about/news/wso2-joins-technology-leaders-in-proposing-oasis-odata-technical-committee/, 2012, (Accessed on 06/2021).

[21] “Oasis launches initiative to standardize rest-based open data protocol (odata) - oasis open,” https://www.oasis-open.org/2012/08/27/odata-tc/, 2012, (Accessed on 06/2021).

[22] L. Ross, “Odata - visualize streaming data the easy way with odata | microsoft docs,” https://docs.microsoft.com/en-us/archive/msdn-magazine/2015/april/odata-visualize-streaming-data-the-easy-way-with-odata, 2015, (Accessed on 06/2021).

[23] R. Cupek and L. Huczala, “Odata for service-oriented business applications: Comparative analysis of communication technologies for flexible service-oriented it architectures,” in *2015 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2015, pp. 1538–1543.

[24] M. Muntean, C. Brândaş, and T. Cîrstea, “Framework for a symmetric integration approach,” *Symmetry*, vol. 11, no. 2, p. 224, 2019.

[25] “Oasis approves odata 4.0 standards for an open, programmable web - oasis open,” https://www.oasis-open.org/2014/03/17/oasis-approves-odata-4-0-standards-for-an-open-programmable-web/, 2014, (Accessed on 06/2021).

[26] W. Redmond, “Technology leaders support oasis standards for open data protocol - stories,” https://news.microsoft.com/2012/05/24/technology-leaders-support-oasis-standards-for-open-data-protocol/, 2012, (Accessed on 06/2021).

[27] P. Cardoso, N. Datia, and M. Pato, “Integrated electromyography visualization with multi temporal resolution,” in *2017 11th International Symposium on Medical Information and Communication Technology (ISMICT)*. IEEE, 2017, pp. 91–95.

[28] M. Kirchhoff and K. Geihs, “Integrating odata services into the semantic web: a sparql interface for odata,” in *Proceedings of the 14th International Conference on Knowledge Technologies and Data-driven Business*, 2014, pp. 1–8.

[29] C. Turbelin and P.-Y. Boëlle, “Open data in public health surveillance systems: A case study using the french sentinelles network,” *International journal of medical informatics*, vol. 82, no. 10, pp. 1012–1021, 2013.

[30] R. Hansen, B. Thomsen, L. L. Thomsen, and F. S. Adamsen, “Smartcampusaau–an open platform enabling indoor positioning and navigation,” in *2013 IEEE 14th International Conference on Mobile Data Management*, vol. 2. IEEE, 2013, pp. 33–38.

[31] W. Lee, B. Priyantha, T. Hart, G. DeJean, Y. Xu, and J. Liu, “The cleo mobile sensing platform,” in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, 2012, pp. 371–372.

[32] J. Lerman, “Msdn magazine: Data points - slice and dice odata with the jquery datatables plug-in | microsoft docs,” https://docs.microsoft.com/en-us/archive/msdn-magazine/2011/

february/msdn-magazine-data-points-slice-and-dice-odata-with-the-jquery-datatables-plug-in, 2011, (Accessed on 06/2021).

[33] "Outsystems to discuss the transformational impact of low-code at gartner application strategies & solutions summit | business wire," https://www.businesswire.com/news/home/20191204005674/en/OutSystems-to-Discuss-the-Transformational-Impact-of-Low-Code-at-Gartner-Application-Strategies-Solution 2019, (Accessed on 08/2021).

[34] "Phoenixdx - transform your ideas into business value. fast." https://phoenix-dx.com/, (Accessed on 09/2021).

[35] "Data visualization | microsoft power bi," https://powerbi.microsoft.com/en-us/, (Accessed on 09/2021).

[36] "Odata documentation - odata | microsoft docs," https://docs.microsoft.com/en-us/odata/, (Accessed on 09/2021).

[37] N. Prat, I. Comyn-Wattiau, and J. Akoka, "Artifact evaluation in information systems design-science research-a holistic view." *PACIS*, vol. 23, pp. 1–16, 2014.

[38] "Postman api platform | sign up for free," https://www.postman.com/, (Accessed on 09/2021).

[39] C. Sells, "Odata and atompub - building an atompub server using wcf data services | microsoft docs," https://docs.microsoft.com/en-us/archive/msdn-magazine/2010/august/odata-and-atompub-building-an-atompub-server-using-wcf-data-services, 2010, (Accessed on 06/2021).

[40] "Core informatics introduces odata api for platform for science," https://www.prnewswire.co.uk/news-releases/core-informatics-introduces-odata-api-for-platform-for-science-596654711.html, 2016, (Accessed on 06/2021).

[41] J. Lerman, "Data points - create and consume json-formatted odata | microsoft docs," https://docs.microsoft.com/en-us/archive/msdn-magazine/2012/july/data-points-create-and-consume-json-formatted-odata, 2012, (Accessed on 06/2021).

[42] D. Fiori and S. Guerrero, *Custom Fiori Applications in SAP HANA*. Springer, 2020.

[43] "Mendix helps enterprises drive digital innovation with new platform release | business wire," https://www.businesswire.com/news/home/20150715005344/en/Mendix-Helps-Enterprises-Drive-Digital-Innovation-with-New-Platform-Release, 2015, (Accessed on 06/2021).

[44] "New capabilities in wso2 open source integration platform enhance service and process orchestration of internet of things applications," https://wso2.com/about/news/new-capabilities-in-wso2-open-source-integration-platform-enhance-service-and-process-orchestration-of-iot-
2015, (Accessed on 06/2021).

[45] S. Iannuzzi, "Odata - odata, the entity framework and micrsosoft azure access control | microsoft docs," https://docs.microsoft.com/en-us/archive/msdn-magazine/2012/october/odata-odata-the-entity-framework-and-micrsosoft-azure-access-control, 2012, (Accessed on 06/2021).

[46] D. Kiely, "Odata in sql server," https://www.itprotoday.com/web-application-management/odata-using-wcf-data-services-access-sql-server, 2013, (Accessed on 06/2021).

[47] A. J. Brust, "Redmond review: Andrew brust likes what he sees with odata – visual studio magazine," https://visualstudiomagazine.com/articles/2010/04/01/open-data-open-microsoft.aspx, 2010, (Accessed on 06/2021).

[48] "Outercurve foundation announces contribution of odata validation project," https://www.prnewswire.com/news-releases/outercurve-foundation-announces-contribution-of-odata-validation-project-12
html, 2011, (Accessed on 06/2021).

[49] "Progress enhances datadirect cloud with odata connectivity - sd times," https://sdtimes.com/progress-enhances-datadirect-cloud-odata-connectivity/, 2014, (Accessed on 06/2021).

[50] D. Esposito, "Cutting edge - queryable services | microsoft docs," https://docs.microsoft.com/en-us/archive/msdn-magazine/2015/april/cutting-edge-queryable-services, 2015, (Accessed on 06/2021).

[51] P. Modderman, C. Goebels, D. Nepraunig, and T. Seidel, *SAP Gateway and OData*. Rheinwerk Publishing, 2019.

[52] C. Goebels, P. Modderman, D. Nepraunig, and T. Seidel, *SAPUI5: The Comprehensive Guide*. Rheinwerk Publishing, 2020.

[53] M. Kirchhoff and K. Geihs, "Semantic description of odata services," in *Proceedings of the Fifth Workshop on Semantic Web Information Management*, 2013, pp. 1–8.

[54] M. Baxter-Reynolds, *The Six Bookmarks Server Service*. Springer, 2010.

A

SLR Obtained Studies Table

Table A.1: List of the obtained studies.

#	Title and Ref	Year
1	A Sensor Data Mediator Bridging the **OGC!** Sensor Observation Service (SOS) and the OASIS Open Data Protocol (OData) [15]	2013
2	APIComposer: Data-driven Composition of REST APIs [16]	2018
3	Build Great Experiences on Any Device with OData [13]	2011
4	Building an AtomPub Server Using WCF Data Services [39]	2010
5	Building Rich Internet Apps with the Open Data Protocol [19]	2010
6	Core Informatics Introduces OData API for Platform for Science [40]	2016
7	Create and Consume JSON-Formatted OData [41]	2012
8	Custom Fiori Applications in SAP HANA : Design, Develop, and Deploy Fiori Applications for the Enterprise [42]	2020
9	Data Marketplaces Have Yet to Deliver on Early Promise [12]	2014
10	Data services [11]	2012
11	Framework for a Symmetric Integration Approach [24]	2019
12	Integrated Electromyography Visualization with Multi Temporal Resolution [27]	2017
13	Integrating OData services into the semantic web: a SPARQL interface for OData [28]	2014
14	Mendix Helps Enterprises Drive Digital Innovation with New Platform Release [43]	2015
15	Model-driven development of OData services: An application to relational databases [14]	2018
16	New Capabilities in WSO2 Open Source Integration Platform Enhance Service and Process Orchestration of Internet of Things Applications [44]	2015
17	OASIS Approves OData 4.0 Standards for an Open, Programmable Web [25]	2014
18	OASIS Launches Initiative to Standardize REST-based Open Data Protocol (OData) [21]	2012
19	OData, the Entity Framework and Windows Azure Access Control [45]	2012
20	OData: Using WCF Data Services to Access SQL Server [46]	2013
21	OData for service-oriented business applications: Comparative analysis of communication technologies for flexible Service-Oriented IT architectures [23]	2015
22	Open data in public health surveillance systems [29]	2012
23	Open Data, Open Microsoft [47]	2010
24	Outercurve Foundation Announces Contribution of OData Validation Project [48]	2011
25	Progress Enhances DataDirect Cloud with OData Connectivity [49]	2014
26	Queryable Services [50]	2015
27	REST-based sensor networks with OData [17]	2014
28	SAP Gateway and OData [51]	2019
29	SAPUI5 : The Comprehensive Guide [52]	2020
30	Semantic Description of OData Services [53]	2013
31	Slice and Dice OData with the jQuery Data Tables Plug-In [32]	2011
32	SmartCampusAAU - An Open Platform Enabling Indoor Positioning and Navigation [30]	2013
33	Tech Leaders Support OASIS Standards for Open Data [26]	2012
34	The CLEO mobile sensing platform [31]	2012
35	The Six Bookmarks Server Service [54]	2010
36	Visualize Streaming Data the Easy Way with OData [22]	2017
37	WSO2 Joins Technology Leaders in Proposing OASIS OData Technical Committee [20]	2012

B

An OData Service Document Listing

Listing B.1: Metadata document of the Northwind project.

```
{
    "@odata.context":"https://phoenixpressservicesptyltd-dev.outsystemscloud.com/CDS/
        rest/odata/Northwind/v1/$metadata",
    "value":
        [
            {
                "name":"Category",
                "kind":"EntitySet",
                "url":"Category"
            }
            {
                "name":"Customer",
                "kind":"EntitySet",
                "url":"Customer"
            }
            {
                "name":"Employee",
                "kind":"EntitySet",
                "url":"Employee"
            }
            {
                "name":"Supplier",
                "kind":"EntitySet",
                "url":"Supplier"
            }
            {
                "name":"Shipper",
                "kind":"EntitySet",
                "url":"Shipper"
            }
            {
                "name":"Product",
                "kind":"EntitySet",
                "url":"Product"
            }
            {
                "name":"OrderDetail",
                "kind":"EntitySet",
                "url":"OrderDetail"
            }
            {
                "name":"Order",
                "kind":"EntitySet",
                "url":"Order"
            }
        ]
}
```

C

An OData Metadata Document Listing

Listing C.1: Metadata document of the Northwind project.

```
<edmx:Edmx Version="4.0">
    <edmx:DataServices>
        <Schema>
            <EntityContainer Name="Container">
                <EntitySet Name="Category" EntityType="Northwind.Category"/>
                <EntitySet Name="Customer" EntityType="Northwind.Customer"/>
                <EntitySet Name="Employee" EntityType="Northwind.Employee"/>
                <EntitySet Name="Supplier" EntityType="Northwind.Supplier"/>
                <EntitySet Name="Shipper" EntityType="Northwind.Shipper"/>
                <EntitySet Name="Product" EntityType="Northwind.Product"/>
                <EntitySet Name="OrderDetail" EntityType="Northwind.OrderDetail"/>
                <EntitySet Name="Order" EntityType="Northwind.Order"/>
            </EntityContainer>
        </Schema>
        <Schema Namespace="Northwind">
            <EntityType Name="Category">
                <Key>
                    <PropertyRef Name="Id"/>
                </Key>
                <Property Name="Id" Type="Edm.Int64" Nullable="false"/>
                <Property Name="CategoryName" Type="Edm.String" Nullable="false"/>
                <Property Name="Description" Type="Edm.String" Nullable="false"/>
                <Property Name="Picture" Type="Edm.Binary"/>
            </EntityType>
            <EntityType Name="Customer">
```

```xml
            <Key>
                <PropertyRef Name="Id"/>
            </Key>
            <Property Name="Id" Type="Edm.String" Nullable="false"/>
            <Property Name="CompanyName" Type="Edm.String" Nullable="false"/>
            <Property Name="ContactName" Type="Edm.String" Nullable="false"/>
            <Property Name="ContactTitle" Type="Edm.String" Nullable="false"/>
            <Property Name="Address" Type="Edm.String" Nullable="false"/>
            <Property Name="City" Type="Edm.String" Nullable="false"/>
            <Property Name="Region" Type="Edm.String"/>
            <Property Name="PostalCode" Type="Edm.String"/>
            <Property Name="Country" Type="Edm.String" Nullable="false"/>
            <Property Name="Phone" Type="Edm.String" Nullable="false"/>
            <Property Name="Fax" Type="Edm.String"/>
        </EntityType>
        <EntityType Name="Employee">
            <Key>
                <PropertyRef Name="Id"/>
            </Key>
            <Property Name="Id" Type="Edm.Int64" Nullable="false"/>
            <Property Name="LastName" Type="Edm.String" Nullable="false"/>
            <Property Name="FirstName" Type="Edm.String" Nullable="false"/>
            <Property Name="Title" Type="Edm.String" Nullable="false"/>
            <Property Name="TitleOfCourtesy" Type="Edm.String" Nullable="false"/>
            <Property Name="BirthDate" Type="Edm.Date" Nullable="false"/>
            <Property Name="HireDate" Type="Edm.Date" Nullable="false"/>
            <Property Name="Address" Type="Edm.String" Nullable="false"/>
            <Property Name="City" Type="Edm.String" Nullable="false"/>
            <Property Name="Region" Type="Edm.String"/>
            <Property Name="PostalCode" Type="Edm.String" Nullable="false"/>
            <Property Name="Country" Type="Edm.String" Nullable="false"/>
            <Property Name="HomePhone" Type="Edm.String" Nullable="false"/>
            <Property Name="Extension" Type="Edm.Int32" Nullable="false"/>
            <Property Name="Photo" Type="Edm.Binary"/>
            <Property Name="Notes" Type="Edm.String" Nullable="false"/>
            <Property Name="ReportsTo" Type="Edm.Int32"/>
        </EntityType>
        <EntityType Name="Supplier">
            <Key>
            <PropertyRef Name="Id"/>
            </Key>
            <Property Name="Id" Type="Edm.Int64" Nullable="false"/>
            <Property Name="CompanyName" Type="Edm.String" Nullable="false"/>
            <Property Name="ContactName" Type="Edm.String" Nullable="false"/>
            <Property Name="ContactTitle" Type="Edm.String" Nullable="false"/>
            <Property Name="Address" Type="Edm.String" Nullable="false"/>
            <Property Name="City" Type="Edm.String" Nullable="false"/>
            <Property Name="Region" Type="Edm.String"/>
            <Property Name="PostalCode" Type="Edm.String" Nullable="false"/>
            <Property Name="Country" Type="Edm.String" Nullable="false"/>
            <Property Name="Phone" Type="Edm.String" Nullable="false"/>
            <Property Name="Fax" Type="Edm.String"/>
            <Property Name="HomePage" Type="Edm.String"/>
        </EntityType>
        <EntityType Name="Shipper">
            <Key>
                <PropertyRef Name="Id"/>
            </Key>
            <Property Name="Id" Type="Edm.Int64" Nullable="false"/>
            <Property Name="CompanyName" Type="Edm.String" Nullable="false"/>
            <Property Name="Phone" Type="Edm.String" Nullable="false"/>
        </EntityType>
        <EntityType Name="Product">
            <Key>
                <PropertyRef Name="Id"/>
            </Key>
            <Property Name="Id" Type="Edm.Int64" Nullable="false"/>
            <Property Name="ProductName" Type="Edm.String" Nullable="false"/>
            <Property Name="SupplierId" Type="Edm.Int64" Nullable="false"/>
            <Property Name="CategoryId" Type="Edm.Int64" Nullable="false"/>
            <Property Name="QuantityPerUnit" Type="Edm.String" Nullable="false"/>
            <Property Name="UnitPrice" Type="Edm.Decimal" Nullable="false"/>
            <Property Name="UnitsInStock" Type="Edm.Int32" Nullable="false"/>
            <Property Name="UnitsOnOrder" Type="Edm.Int32" Nullable="false"/>
            <Property Name="ReorderLevel" Type="Edm.Int32" Nullable="false"/>
            <Property Name="Discontinued" Type="Edm.Boolean" Nullable="false"/>
            <NavigationProperty Name="Supplier" Type="Northwind.Supplier" Nullable="false"
                 ContainsTarget="true"/>
            <NavigationProperty Name="Category" Type="Northwind.Category" Nullable="false"
                 ContainsTarget="true"/>
            </EntityType>
            <EntityType Name="OrderDetail">
                <Key>
                    <PropertyRef Name="Id"/>
                </Key>
                <Property Name="Id" Type="Edm.Int64" Nullable="false"/>
                <Property Name="OrderId" Type="Edm.Int64" Nullable="false"/>
                <Property Name="ProductId" Type="Edm.Int64" Nullable="false"/>
                <Property Name="UnitPrice" Type="Edm.Decimal" Nullable="false"/>
                <Property Name="Quantity" Type="Edm.Int32" Nullable="false"/>
                <Property Name="Discount" Type="Edm.Decimal" Nullable="false"/>
                <NavigationProperty Name="Order" Type="Northwind.Order" Nullable="false"
                     ContainsTarget="true"/>
                <NavigationProperty Name="Produto" Type="Northwind.Produto" Nullable="false"
                     ContainsTarget="true"/>
            </EntityType>
            <EntityType Name="Order">
                <Key>
                    <PropertyRef Name="Id"/>
                </Key>
                <Property Name="Id" Type="Edm.Int64" Nullable="false"/>
                <Property Name="CustomerId" Type="Edm.String" Nullable="false"/>
                <Property Name="EmployeeId" Type="Edm.Int64" Nullable="false"/>
                <Property Name="OrderDate" Type="Edm.Date" Nullable="false"/>
                <Property Name="RequiredDate" Type="Edm.Date" Nullable="false"/>
                <Property Name="ShippedDate" Type="Edm.Date"/>
                <Property Name="ShipVia" Type="Edm.Int64" Nullable="false"/>
                <Property Name="Freight" Type="Edm.Decimal" Nullable="false"/>
                <Property Name="ShipName" Type="Edm.String" Nullable="false"/>
                <Property Name="ShipAddress" Type="Edm.String" Nullable="false"/>
                <Property Name="ShipCity" Type="Edm.String" Nullable="false"/>
                <Property Name="ShipRegion" Type="Edm.String"/>
                <Property Name="ShipPostalCode" Type="Edm.String"/>
                <Property Name="ShipCountry" Type="Edm.String" Nullable="false"/>
                <NavigationProperty Name="Customer" Type="Northwind.Customer" Nullable="false"
                     ContainsTarget="true"/>
                <NavigationProperty Name="Employee" Type="Northwind.Employee" Nullable="false"
                     ContainsTarget="true"/>
                <NavigationProperty Name="Shipper" Type="Northwind.Shipper" Nullable="false"
                     ContainsTarget="true"/>
            </EntityType>
        </Schema>
    </edmx:DataServices>
</edmx:Edmx>
```