An evaluation of Graph Databases and Object-Graph Mappers in CIDOC CRM-compliant digital archives

LÁZARO COSTA*, NUNO FREITAS*, and JOÃO ROCHA DA SILVA, INESC TEC and Faculdade de Engenharia da Universidade do Porto, Portugal

The Portuguese General Directorate for Book, Archives and Libraries (DGLAB) has selected CIDOC CRM as base for its next-generation digital archive management software. Given the ontology foundations of the CRM, a graph database or a triple store were seen as the best candidates to represent a CRM-based data model for the new software. We thus decided to compare several of these databases, based on their maturity, features, performance in standard tasks and, most importantly, the Object-Graph Mappers (OGM) available to interact with each database in an Object-Oriented way. Our conclusions are drawn not only from a systematic review of related works but from an experimental scenario. For our experiment, we designed a simple CRM-compliant graph designed to test the ability of each OGM/database combination to tackle the so-called "Diamond-problem" in Object-Oriented Programming (OOP), to ensure that property instances follow domain and range constraints.

Our results show that 1. ontological consistency enforcement in graph databases and triplestores is much harder to achieve than in a relational database, making them more suited to an analytical rather than a transactional role, 2. Object-Graph Mappers are still rather immature solutions and 3. neomodel, an OGM for the Neo4j graph database, is the most mature solution in the study as it satisfies all requirements, although it is also the least performing.

CCS Concepts: • **Information systems** → **Digital libraries and archives**; **Network data models**; **Database performance evaluation**.

Additional Key Words and Phrases: Object-Graph Mapping,Graph Databases,Digital Archives, CIDOC CRM,Comparison

ACM Reference Format:
Lázaro Costa, Nuno Freitas, and João Rocha da Silva. 2021. An evaluation of Graph Databases and Object-Graph Mappers in CIDOC CRM-compliant digital archives. *ACM J. Comput. Cult. Herit.* 1, 1, Article 1 (September 2021), 20 pages. https://doi.org/10.1145/3485847

1 INTRODUCTION

The CRM (Conceptual Reference Model) is a model developed by CIDOC (ICOM International Committee for Documentation), a documentation section of the International Council of Museums. In 2014, CRM was formalized as an ontology that includes OOP concepts such as class hierarchies, domains and ranges. This

*All authors contributed equally to this research.

Authors' address: Lázaro Costa, lazaroosta@hotmail.com; Nuno Freitas, nuno.freitas96@gmail.com; João Rocha da Silva, joaorosilva@gmail.com, INESC TEC and Faculdade de Engenharia da Universidade do Porto, Campus da Faculdade de Engenharia da Universidade do Porto, Porto, Porto, 4200-465, Portugal.

defines a modular model while allowing the implementer to control the specificity of each implementation [28]. Recently, EPISA [1] projects were launched in Portugal involving INESC TEC (Institute for Systems and Computer Engineering, Technology and Science), DGLAB (General Directorate for Book, Archives and Libraries) and the University of Évora.

One of the goals of this project is to propose a data model for a new version of the digital archive management software Digitarq. The data model proposal for the new system is a combination of CIDOC CRM and other ontologies [25] implemented over a graph database.

Currently, there are few implementations of the CRM as the core data model for software applications; it is more commonly used to format data as Linked Open Data [21, 38] for interoperability. This is due to the fact that the CRM can be quite complex and represents its data in the form of a graph, for which relational databases are not the best option.

It was decided to implement the digital archive using a graph database, as it is more adequate for representing resources of many types, and where one of the more relevant requirements is the exploration of the relationships between them. Graph databases offer greater freedom and scalability for the representation of data models where there are many types of resources and relationships. In relational databases, referential integrity, primary keys and other constraints are used to enforce the database schema, which ensures the validity of the data model. Graph databases, on the other hand, typically delegate the responsibility of ensuring the validity of the data model on the application layer. Graph databases can be used to represent ontologies, and ontological consistency is often validated using engines called reasoners. Reasoners are able to enforce the concepts defined in an ontology, such as Class hierarchies, Domains and Ranges, cardinalities of Properties, as well as Axioms, making sure that there are no contradictions in the ontology. However, these validations require inference, which implies a heavy computational price, making such validations impractical when the ontology is under constant modification.

For simpler ontologies, basic constraints such as the Domain, Range and cardinality of Properties are enough to ensure a consistent ontology. This can be achieved through the use of OGM. OGM are mapping frameworks for graph databases [11]. They are intended as an abstraction layer that makes it easier for programmers to manipulate the entities and properties of the graph. They map database concepts to classes and instances (in terms of OOP), that can be manipulated programmatically to interact with the database. When compared to a reasoner, these frameworks can enforce simpler constraints, but are able to do so at a much lower computational price. When the programming language allows (e.g. Java or Python), OGM enable the definition of rules, directly in the source code, as properties, methods and class annotations. This improves the detail of model specification while keeping the code clean and easy to read. Boilerplate code necessary to perform simple CRUD (Create, Read, Update and Delete) operations on class and property instances is also greatly reduced. Despite the promise of functionality and clean code, we found that OGM maturity and feature completeness can vary greatly. There were also no systematic evaluations of these solutions as far as we know, so a prototype had to be developed to determine which solution was more suited to the needs of the ICON and EPISA projects.

A graph database can be used as an Online Transaction Processing (OLTP) system or loaded periodically with the result of an ETL (Extraction, Transformation and Loading) process from a relational database

[1] https://www.inesctec.pt/en/projects/episa

that in turn assumes the role of the OLTP system. In the latter scenario, the validation of the graph's ontological consistency is performed during the ETL, a process that should run periodically without hurting the availability of the transactional system. For a transactional system supported by a graph database, however, the OGM or the database need to perform validations whenever changes are made to the graph, while keeping the system available.

We present a comparison of OGM solutions to support graph databases and ontologies for the core data model of a transactional archival software system. Our data model is based primarily on ArchOnto—an ontology being developed as the data model for the digital archive [26]. However, since graph databases are flexible, other ontologies can be used in combination with ArchOnto, as long as their concepts are modeled using the OGM. This makes this comparison relevant for any organization considering the implementation of a CIDOC CRM based software solution for their digital archive.

This paper starts with two sections of literature review. Section 2 with a broader scope, is a short comparative analysis between relational and non-relational databases, necessary to justify our choice of non-relational when implementing a CIDOC CRM-based data model. Section 3 details the base requirements that we gathered for the new DGLAB digital archive and analyses previous works within their scope. Section 4 describes how we designed an experiment to compare four OGM/Database technology stacks according to their compliance with our requirements. Section 5 presents the results of that experiment, and Section 6 presents conclusions and future work.

2 RELATIONAL VS. NON-RELATIONAL DATABASES FOR A CIDOC CRM-BASED DATA MODEL

Organizations that store large volumes of non-structured data are increasingly adopting non-relational databases, or NoSQL databases, mainly because of their better scalability in cloud contexts [24, 29, 34].

Non-relational databases differ from relational ones in multiple aspects; their main focus is the analysis and processing of a large volume of data, offering scalability and storage and application computing requirements such as Big Data Analytics [42], Business Intelligence [43] and social networking [27]. Cassandra, BigTable, CouchDB, Project Voldemort, and Dynamo, all store large volumes of data and are all of them projects that use non-relational databases, that is, NoSQL projects [49].

The structuring of their data does not use tables to store information, they do not use SQL as a querying language and they do not always offer ACID (Atomicity, Consistency, Isolation, Durability) transaction properties, in exchange for higher scalability and raw performance [1, 23, 49]. Most non-relational databases are available as open source solutions [23]. While this is positive in terms of freedom and reduced licensing costs, support and regular maintenance are both dependent on community contributions [15, 23].

Beyond this, there is also the problem of non-relational databases being disk-based and thus implementing buffer pools and multi-threading, but in the moment when transactions occur it is necessary to block the buffer, which can lead to performance decreases [23]

Non-relational databases may be classified according to their data organization. We describe with more detail some of the elements of this classification [23]:

- Document Store, also known as "Document Oriented Database", allows the encapsulation of flexible attribute groups, represented by a document, in the form of standard formats like XML, BSON and PDF. Each document is represented by a unique key in string (URI or path).

- Graph Databases, are databases that do not use data schemas to represent information, and instead use a graph structure of data, containing nodes, edges and properties to represent information. With this it is possible to explore the relationships between the various resources of the graph.
- XML Databases, are database management systems used to represent data in the XML format. By using XSL stylesheets it is possible to transform an XML document from one structure into another [2, 23]. This has been used in RiC-O, an open-source mapping tool that facilitates the batch conversion of XML documents containing ISAD (International Standard Archival Description) records into their RiC (Records In Context) counterparts. RiC is an OWL-2 ontology for describing archival record resources [2].

Beyond the classification previously mentioned there are also other classifications to non-relational databases, namely: Key Value Stores, Column Oriented Databases, Object Oriented Databases, Grid and Cloud Databases, Multidimensional Databases, Multivalue Databases, and Multimodel Databases [24].

Relational databases have been traditionally associated to the ACID properties, while non-relational models are typically associated with the BASE (Basically Available, Soft state, Eventual consistency) properties, which opt for eventual consistency instead of enforcing immediate consistency. Eventual consistency does not guarantee, for example, that all reads after a write return the same value until the database state converges at some instant in the future.Non-relational databases focus on permanent availability instead [36].

As the number of classes in a data model increases, so does the number of tables in its corresponding relational model. As more classes are added, so does the number of connecting tables necessary to represent edges between the instances of those classes ("many-to-many" associations), making such an approach impractical in the long term. On the other hand, graph databases are especially designed to handle the requirements of heavily connected data models.

The CIDOC CRM, initially developed to homogenize museum inventory databases, became an international standard for cultural heritage in 2006 and at the moment offers multiple extensions. The CRM and its compatible models are able to handle diverse and incomplete information, with a vast number of properties and classes modeled. Most properties are also optional and have “many-to-many” cardinality.

Because the CRM is formalized as an ontology, it is easy to combine with other ontologies, such as to GEOSPARQ CRMgeo, an ontology that integrates space-time properties of CIDOC CRM items, introduced to separate real world classes from information classes. This distinction between the real word and the world described by information is related only to the time and geometry dimensions [48].

3 REQUIREMENTS IN GRAPH TRANSACTIONAL SYSTEMS

The Semantic Web [5] offers the ability to infer new knowledge from existing facts, but such operations rely on maintaining the consistency of the knowledge base as information is added or removed. In particular, information inferred from statements that are erased in the meantime must also be removed [6].

The growth of the so-called Web of Data and the wider adoption of Linked Open Data (LOD) prompted the creation of applications based on Resource Description Framework (RDF) [3] data models that consider data versioning, ontology versioning and synchronization of information from different systems [41].

[2] https://ica-egad.github.io/RiC-O/about.html
[3] https://www.w3.org/RDF/

Applications of ontologies in operational systems are still few when compared to their relational model counterparts. However, attempts been made to use ontologies to tackle common requirements such as synchronization and versioning. A study was carried out in order to create a triplestore storage based on servers and clients, and synchronize those changes between the various environments [8]. In addition, an approach was proposed for distributed knowledge bases versioning based on the RDF data model with support for ontologies in constant evolution, i.e. versioned [4].

A new digital archive management system must handle frequent and concurrent updates to the database, as it is intended to support the day-to-day activities of the archivists from DGLAB. However, non-functional system requirements include the use of the CIDOC CRM ontology in the transactional system. A relational model was quickly ruled out due to the high complexity and high degree of connectivity between resources of the ontology [33].

Throughout this section we will cover four requirements that we consider to be essential for the new CRM-based digital archive software of DGLAB. Interoperability is required to make sure that the data can be queried and interpreted by external systems, ontological consistency of the graph must be enforced whenever modifications are performed, versioning should be available for audit purposes, and finally, the programming ecosystem should provide adequate mapping tools and connectors to ensure a modular software architecture.

3.1 Interoperability

RDF was designed by the World Wide Web Consortium (W3C) [4] as the standard model for conceptual descriptions and data modeling in web resources.

A triplestore or RDF store aims at the representation of triples in order to allow their recovery through semantic queries. A triple follows a Subject-Predicate-Object structure, that can represent all kinds of information [7].

Graph databases offer good performance in applications where relationships between entities are essential [31]. They provide ACID properties and disk persistence [31]. Some graph databases also use a relational database as the underlying storage and engine processing that allows for the serialization of the graph data [40].

Graph databases can offer better efficiency and allow for more freedom in the creation of data models than a triple store [2]. Their main data model, the Labeled Property Graph (LPG) [44] allows model designers to add properties to the nodes and edges themselves, instead of having to add new triples for representing those properties. LPG allows for a more compact and rich representation of qualified relationships as can be seen in Figure 1.

Conversely, a triple store is often the best candidate database system when implementing a data model based on ontologies. This is because they are designed to represent relationships between resources in a graph and can also be connected to an inference engine to uncover new knowledge. They implement the SPARQL (SPARQL Query Language for RDF) language, a query standard recommended by the W3C and assume the adoption of defined concepts in ontologies [20]. This way, they can be considered more interoperable than

[4] https://www.w3.org/

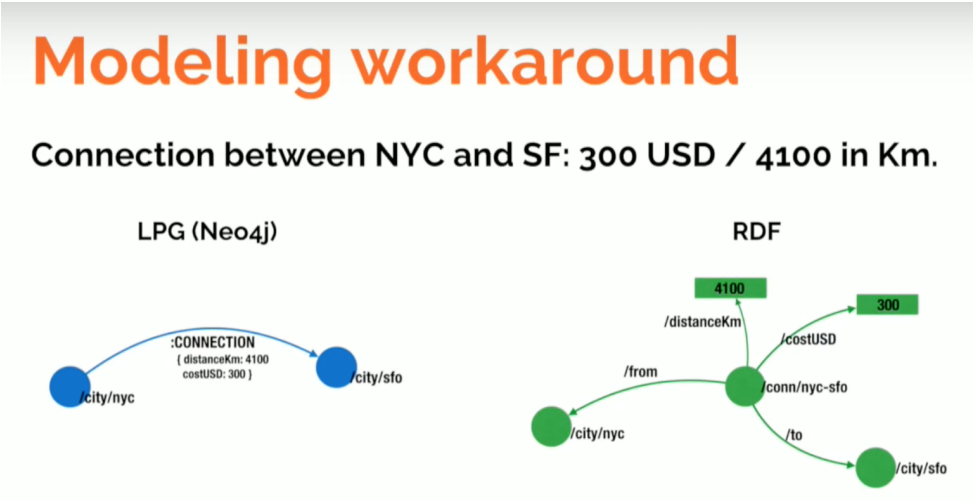

Fig. 1. Representing the same graph using LPG vs RDF. Image by Neo4j [35].

graph databases, which tend to implement manufacturer-specific query languages and add to support for more standard languages such as Gremlin [5] or GQL [6] as connectors or additional APIs [31].

As a downside to this flexibility and interoperability, triple stores tend to be more adequate for WORM (Write-Once-Read-Many) applications instead of a viable alternative for OLTP systems. This is mainly because of their locking model, which makes them too slow to support transactional systems [35].

3.2 Enforcement of ontological consistency

Updating data in a graph database or triple store can easily introduce semantic inconsistencies or eliminate links between resources of the graph, which leads to the separation of entire parts of the whole (subgraphs) [30]. This contrasts with relational databases, where referential integrity, primary keys and other constraints can be used to efficiently enforce a database schema and thus maintain the consistency of the data model.

There are fewer applications reliant on ontologies for OLTP systems, as the price of enforcing the ontological consistency of the knowledge graph after every modification is very expensive in computational terms. Conversely, the adoption of a graph database in an OLAP (Online Analytical Processing) system is viable. While validating the consistency of the graph can take a long time, such operations are only performed periodically and in well-defined moments and the system can thus stop responding to requests while those updates are running. Such graph consistency validations in an OLTP system are not a viable option, since they would block the system for a long time during the validation process [8], and an OLTP system should respond within a matter of milliseconds.

Some triple stores and graph databases provide support for plugging in an inference engine such as FaCT++ [46], Pellet [45] and others. OWL API [19] is one such solution, proposing an easy to use Java API

[5] https://tinkerpop.apache.org/gremlin.html
[6] https://www.gqlstandards.org/

for working with OWL and RDF. The Apache Jena [7] library also offers inference support and consistency checking interfaces [32]. In alternative, some of these also offer rule-based consistency validation.

OpenLink Virtuoso [8], a triple store, relies on a pre-determined set of queries to be ran whenever graphs are modified, taking advantage of some inference rules [9]. It supports inference over some properties through rule sets [10]. From version 5.00.3031, it correctly interprets `rdfs:subClassOf` and `rdfs:subPropertyOf` properties for triple inference. According to the database documentation, `owl:sameAs` is also considered for arbitrary subjects and objects if specially enabled by a pragma in the query. `owl:sameAs`, `owl:equivalentClass` and `owl:equivalentProperty` are also considered when determining subclass or subproperty relations. If two classes are equivalent, they share all instances, subclasses and superclasses directly or indirectly stated in the data for either class [17]. Other RDF Schema or OWL information is not taken into account [11].

In the case of Neo4j [12] and GraphDB [13] (both graph databases) consistency validation can also be performed via rule sets that are ran against the graph upon modification. These rule sets can be specified using SHACL (Shapes Constraint Language), a W3C recommendation [9].

Studies on the performance of graph databases in consistency validation tasks are harder to find than those designed around tasks such as graph traversal, node centrality or graph diameter, which are more readily available [10, 31]. The lack of diversity in synthetic benchmarks targeting graph databases has also been identified in a recent overview [13], indicating that perhaps consistency checks should be included in more of these benchmarks, given the importance of the task in real-world scenarios.

This relative lack of benchmarks covering the validation of ontological consistency has been reported in the OWL API open-source project, for example [14]. The responses of the developers hinted at *possibly* linear time and space requirements for consistency validations. In 2015, a study concluded on linear time complexity with the number of triples when performing a consistency validation [32], thus matching the estimates of the OWL API developers. These authors also stated that, in their system, the time spent to run the consistency check is around 10s for a 5 million triple graph and reaches approximately 50s for a graph consisting of 27 million triples. Given these numbers, if such checks were to be performed every time a user performs a modification to the graph (OLTP system) such a system would be unusable.

3.3 Data Versioning

The ability to keep an audit trail of modifications is a relevant requirement when constructing an archive management system. To achieve this, it is necessary to save records of modifications made to metadata of documents, collections or fonds. Eventual changes to the relationships between entities registered in the system also need to be versioned (for example, moving a document from one collection to another).

In relational databases, audit trails can be kept using auxiliary tables that contain the past values of records in the original table, and so the granularity of the versioned registers is clear. In a graph database or triple store, however, it is more difficult to establish the granularity of the changes as it becomes necessary

[7] https://jena.apache.org/
[8] https://virtuoso.openlinksw.com/
[9] https://community.openlinksw.com/t/validation-of-rdf-graphs-consistency/1942/4
[10] http://docs.openlinksw.com/virtuoso/rdfsparqlrulesubclassandsubprop/
[11] http://docs.openlinksw.com/virtuoso/rdfsparqlruleintro/
[12] https://neo4j.com/docs/labs/nsmntx/current/validation/
[13] https://graphdb.ontotext.com/documentation/free/shacl-validation.html#supported-shacl-features
[14] https://github.com/owlcs/owlapi/issues/730

to save copies of subgraphs of different dimensions. For example, it may be necessary to save the past values of a datatype property object (relatively simple) but it might also be necessary to save the history of changes of the relationships between two resources (requiring a copy of an entire subgraph). This problem of varying change granularity is far from trivial, and data versioning needs to be designed in a case by case basis.

In the context of triple store versioning, it is first necessary to analyze the granularity of the alterations, as for a RDF statement to be changed it is necessary to perform a removal operation followed immediately by the addition of a new RDF statement [8, 37]. Another work focused on keeping track of the different versions of the ontologies in other to enable ramification and fusion operations between the various versions [4].

3.4 Programming ecosystem

The integration and interaction between the database and the business logic is crucial when selecting a technology stack for any software solution. The code should be expressive and modular, with a minimum of hand-written queries, since those make the solution brittle and hard to adapt to changes in the data model. Given the widespread adoption of OOP, it should adopt frameworks that enable the mapping of graph entities and properties into instances and classes (in OOP terms). This encourages code organization which improves ease of modification by external developers—an important aspect for open-source projects [18] such as the new version of this new digital archive system.

4 A HANDS-ON, EXPERIMENTAL COMPARISON BETWEEN OGM FRAMEWORKS

Using ontologies it is possible to define domains and ranges of properties as well as their permissible cardinality. The same does not always carry over when using OGM frameworks, even though some allow for the validation of domains and ranges. Most OGM are still underdeveloped in regard to cardinality validation [12, 47].

We selected several OGM and corresponding database management systems with the goal of determining which of them could consistently represent the CRM. These are: GVerse, using the DGraph database, ArangoJS and OrangoJS for ArangoDB, and the Neo4j OGM and Neomodel for interacting with Neo4j. The main goal of this experiment is to determine which of these are able to satisfy 3 basic requirements: handling the "Diamond Problem" in Object-Oriented programming (OOP) and enforcing property domains and ranges as well as their cardinalities.

To achieve our goal, we designed a small CRM-compliant graph to test the alternatives under analysis. Figure 2 represents this graph, including all the nodes and property instances that should be allowed (i.e. correct according to the CRM) and those that should be forbidden, as they would introduce inconsistencies as per the CRM.

4.1 Tackling the "Diamond Problem"

In OOP, the "Diamond Problem" arises whenever there is a set of classes that inherit from a parent class while also being superclasses of a third one. This is a multiple inheritance pattern—in other words, a grandchild class should have properties and methods inherited from any of its parents, as well as from its grandparent.

In order to compare the various OGM, we used a small subset of CRM entities and properties to build a graph representing the Eiffel Tower and the Tokyo Tower, which shares some of its features. This is meant

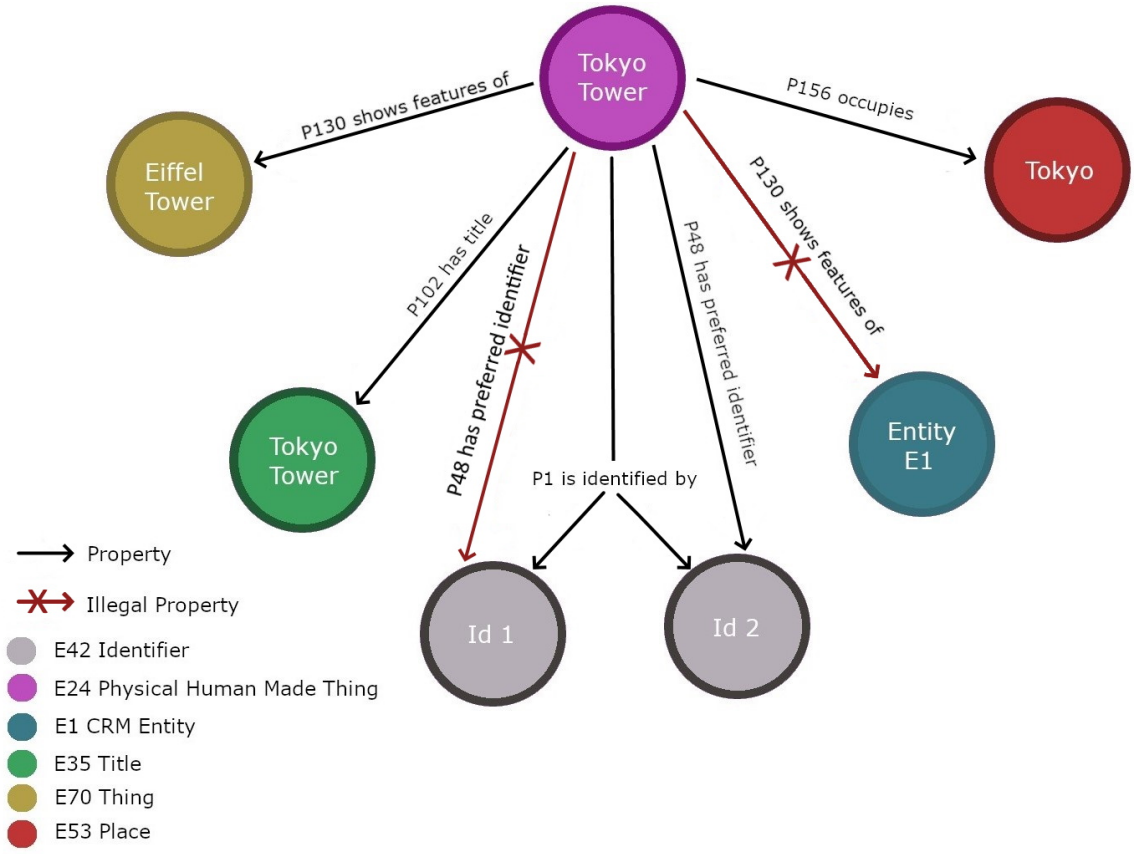

Fig. 2. Data model for the experiment.

to test how each OGM handles the Diamond Problem raised by multiple inheritance in OOP, as well as the validation of the Domains, Ranges and cardinalities of properties.

As an example using the CRM, we present a class hierarchy in Figure 3. The Tokyo Tower (an `E24 Physical Man Made Thing`, subclass of `E70 Thing`) is connected to the Eiffel Tower through an instance of `P130 shows features of`, a property with `E70 Thing` as its domain. Also, `E24 Physical Man Made Thing` is a subclass of both `E71 Man Made Thing` and `E18 Physical Thing` and `E70 Thing` is a superclass of them both (visible in the right side of Figure 3).

In order to pass our requirements, an instance of `P130` is created correctly if the OGM or programming language it utilizes is capable of handling the Diamond Problem.

4.2 Domain and range enforcement

In order to test if the OGM correctly creates relationships with a correct domain and range, we created two entities, a `E35 Title` and a `E53 Place`, named Tokyo and Tokyo Tower respectively. They are connected to the `E24 Physical Human Made Thing` Tokyo Tower, through an instance of `P156 occupies` and another

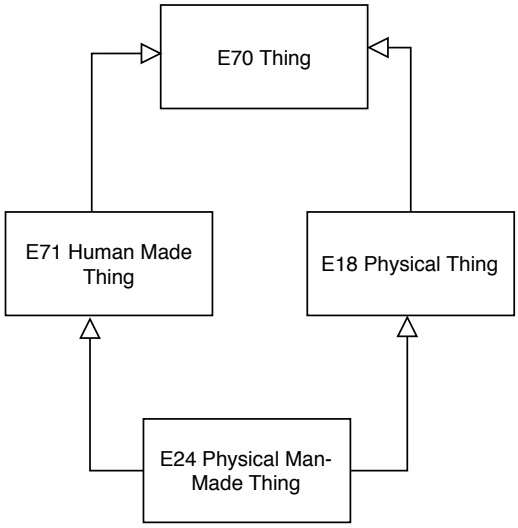

Fig. 3. A depiction of the "Diamond Problem" using UML (Unified Modelling Language) syntax.

instance of `P102 has title`. The successful creation of these relationships in the graph database through the OGM validates the capability to create relationships successfully, and is a way to demonstrate how a specific entity would normally be connected to other concepts through the CIDOC CRM.

To test if the OGM validates ranges and domains, we attempt to connect an instance of `E24` to another of `E1` through a `P130 shows features of`. This is an illegal operation due to the range of `P130`, which in the CRM is not `E1`. If the OGM creates it anyway, then we consider it to fail the experiment as it does not properly handle Domain and Range restrictions on properties.

4.3 Cardinality enforcement

In order to test if the OGM and graph database are capable of enforcing cardinality restrictions, two `E42 Identifier` entities are created, one named `ID1` and another is named `ID2`. These are meant to abstractly represent two differing identifiers that a system might give an entity (often found when identifying the same resource in different contexts).

They are both connected via an instance of `P1 is identified by` from the Tokyo Tower `E24` entity (which is valid, because P1 has a "one to many" cardinality). The experiment scenario also attempts to create an instance of `P48 has preferred identifier` from and to the same entities as the previous property (which should fail, as `P48` has a "one to one" cardinality).

5 EXPERIMENT RESULTS

In this section we present the results of the implementation of the data model described in Section 4 using four different technology stacks. We compare each stack according to four dimensions: its ability to enforce ontological consistency, the expressiveness of the resulting code, the level of maintenance by their communities and the degree of adoption of the underlying database.

The analysed OGM and the versions used in this experiment can be seen in Table 1. To complement the comparison between the OGM, we also present some source code excerpts. For brevity's sake, we selected a subset of the target model, highlighted in Figure 4.

This subset includes the modeling of the `P130 shows features of` property, which has `E24 Physical Human Made Thing` as its domain and range. The instantiation of the Eiffel Tower and the Tokyo Tower nodes (both instances of `E24`) is also highlighted in every source code excerpt, as well as the creation of an instance of `P130` to establish the relationship between them.

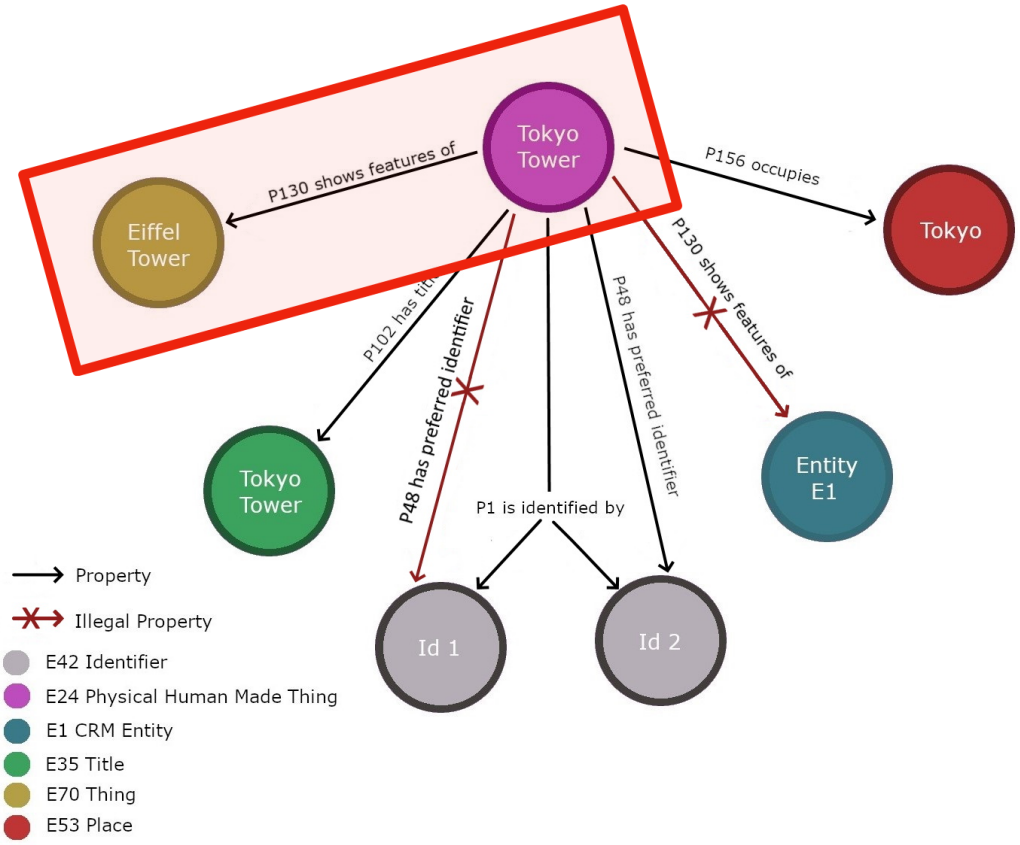

Fig. 4. Subset modeled in source code examples.

Table 1. All tested Frameworks

OGM/Dimension	GVerse	OrangoJS	ArangoJs	Neo4j OGM	Neomodel
Graph Database	DGraph	ArangoDB	ArangoDB	Neo4j	Neo4j
OGM Version	0.4.5	1.0.0.beta-1.1	6.14.1	3.2.17	4.0.1

5.1 GVerse

GVerse [15] is an open-source OGM library designed for DGraph, a "high-performance, open-source graph database". It is "written in Typescript and Typescript 3 and supports ECMAScript 6".

The positive points of this OGM start with the programming language. Typescript is a strongly-typed language that is transpiled into JavaScript by the Typescript compiler. This makes it very flexible, as one can combine the vast collection of libraries written in JavaScript with strong typing that reduces bugs from type mismatches, which can frequently happen in JavaScript applications. Another advantage for programmers for TypeScript when compared with JavaScript is effective TypeScript auto-complete functionality in every major IDE.

To test GVerse 0.4.5 with DGraph, we deployed a Docker container with DGraph 1.2.2 and established a connection as described in the GVerse documentation. While executing the code described in the tutorial and brief documentation published by the library creators, we encountered `Unimplemented Exceptions`—which indicates that some of the basic features were left unfinished. Running the tests would prove unfruitful, with 3 out of 5 integration tests failing. An analysis of the few tests revealed poor coverage.

Given these poor results we rejected this library, even though it seemed very promising at first. No conclusions could be taken about its support for domain and range validations when editing edges, nor about its ability to enforce property cardinality constraints.

5.2 OrangoJS

OrangoJS [39] is another open-source OGM [16] and ORM that allows the creation of data schemas, automate the creation of collections and indexes, as well as execute queries to the database. This OGM connects to ArangoDB [3] databases. This OGM, has the advantage of being capable of defining a schema with the proper validation for a given collection in an intuitive way.

OrangoJs allows for the creation of edges between collections as well as inserting validations for a given collection in an intuitive manner.

However, the OGM does not have any kind of validation regarding neither the cardinality of the edges nor the domain and range of these. When an edge is created it is possible to add more than a single entity to the domain and range, but there is no type of information one can add to restrict the cardinality of the domain and range of the edges.

Beyond that it is impossible to extend the schemas of the collections, and this way it is necessary to create a schema for each collection even in the case of the majority of the attributes coinciding [17].

Due to the lack of validation of domains and ranges, cardinality, as well as the fact that it is not possible to use the arangodb's graph features, means OrangoJs in general cannot fill the basic requirements to map

[15] https://github.com/gverse/gverse
[16] https://github.com/roboncode/orango
[17] https://orango.js.org/guide/working-with-models.html#defining-a-model

the CIDOC CRM. In addition, OrangoJs has been discontinued and is no longer being maintained (see Appendix A).

5.3 ArangoJS

ArangoJS [18] is an open-source JavaScript driver created for the graph database ArangoDB. The ArangoDB's database is a conjoined set of Documents, which are database objects identified by a Document Handle, and in a graph a baseline Document would is a node.

In ArangoDB graphs, relationships between nodes are expressed as a different type of Document named "Edge". Edges differ from normal documents as they must always have two specific properties `"_from"` and `"_to"`, which describe the nodes that the edge connects, the tail and head of the edge, respectively.

The organization of an ArangoDB database relies on Collections, which are sets of documents, identified by a collection identifier. It is possible to create collections of the two types of ArangoDB documents, baseline document collections, and edge collections.

When creating graphs, ArangoDB uses Edge Definitions in order to be able to define which Edge Collections may be used in a graph, and is the only way to properly designate a domain and a range for edges, as Edge Definitions may establish special `"_from"` and `"_to"` properties for the Edge Collections that define Document Collections as domains and ranges for the edges. Without these, all graphs are undirected. Due to the fact that CIDOC CRM graphs are directed, the use of Edge Definitions is imperative. However, it is impossible to define cardinality for the edges.

In order to construct the experiment with ArangoJS each node and edge type utilized was defined as a class mix-in methods were utilized to build the class hierarchy present in CIDOC CRM. From our experiment we were able to confirm that the use of mix-in methods can create the multiple hierarchy necessary to create the CIDOC model. Mix-in methods solve the Diamond Problem by prioritizing conflicting methods that are inherited by the last declared mixed-in class.

Despite ArangoJS being capable of technically being able to define CIDOC CRM graphs, the ArangoJS driver is incapable of checking the restrictions on ranges and domains of the edge definitions (despite being able to create edge definitions). Because of this, the driver can insert edges that are erroneous to the model; this coupled with the fact that ArangoDB has no way of controlling cardinality makes it less useful when attempting to build a CIDOC CRM data model (see Appendix B).

5.4 Neo4j OGM

The Neo4j OGM [19] is an open-source Object Graph Mapper designed for Java, that maps graph entities from a Neo4j Database into Java objects. Neo4j itself is a graph database that organizes both its nodes and edges through the use of tags and indexes. The tags allow they typing of nodes and edge, which are the main tool in the mapping of java objects. The Neo4j database is a directed graph database, however there is no way to be able to define domains and ranges for different types of edges. As such, in order to be able to define the CIDOC model, all verifications need to be programmed manually. In a similar vein cardinality is not something that can be defined nor verified in the graph itself [12, 47].

[18] https://github.com/arangodb/arangojs
[19] https://github.com/neo4j/neo4j-java-driver

In order to test the Neo4j OGM and be able to model CIDOC CRM it was necessary to use Apache Groovy, a language built on the Java platform, that enables all of the OGM's features. This is because Java does not handle the Diamond Problem in a way that allows the creation of multiple classes. Instead, the only way to handle this is the creation of interfaces and having them apply multiple inheritance. As interfaces are stateless, this would require unsustainable amounts of code repetition. The Apache Groovy solution to the Diamond Problem is preferable, as it utilizes objects named `Traits`, a special type of interface that is stateless and doesn't require the repetition and coding of every method mentioned in the interface, instead being able to be coded on the trait and inherited directly. The Diamond Problem is solved by prioritizing conflicting methods from the last (in order of writing) inherited trait in the class specification.

The main issues with the this OGM is there is no automatic way of defining domains and ranges, and verification of the integrity of the model still needs to be constructed and performed in every change of the graph. The same situation applies for cardinality constraints. In addition, the use of traits creates a software bug where not all necessary tags are added to a Neo4j node. Tags specify the classes that a Neo4j node is an instance of, according to the LPG model—a failure to properly record such tags would imply an inconsistent data model.

5.5 Neomodel

NoSQL Database Neomodel [20] is an open-source OGM for Neo4j as well, developed for Python. Neomodel, was developed as an independent project from the official neo4j OGM, however it is built on top of the official Neo4j Python driver [21]. Python has the capability of handling the Diamond Problem by its own design. In order to avoid it and allow for the use of multiple inheritance of objects, it linearizes the order of inheritance by the order specified when the class is defined, with the first class called having priority, then the next one and only after the parent classes [14].

Neomodel's features include the capability of setting domains and ranges for specific types of edges, and is capable of validating these when new edges are inserted or modified without need for creating other methods of validation. In a similar manner it is possible to define cardinality for each edge type and Neomodel validates this without outside functions being necessary. Our tests using Neomodel determine that it is able to satisfy our consistency validation requirements and as such, we were able to model the vast majority of CIDOC CRM using this OGM.

In addition to the features necessary for CIDOC CRM modelation Neomodel has in its repertoire multiple functions to perform more advanced queries to the database without being necessary to write any Cypher (the Neo4j querying language). These include node comparison, property analysis and complex look-ups. In addition it is possible to perform operations supported by regex to perform other searches and analysis if necessary (see Appendix C).

5.6 Hardware and software configuration

An experiment was created in order to compare the efficacy of the 3 OGM when subject to overload. The experiment consists in the creation of the graph of the Figure 2 and erase it, and repeat this process 1000

[20] https://github.com/neo4j-contrib/neomodel
[21] https://neomodel.readthedocs.io/

and 10000 times. This process was executed 5 times, and then the average of the experiment duration is calculated for each OGM.

The system utilized to execute the experiments was Ubuntu Linux, version 18.04. The machine used possesses a Intel Core i7-8750HQ with 6 cores and 12 threads at 2.20 GHz with 16GB of RAM.

At the moment of the experiment's execution, only the necessary programs for its active functioning were being executed.

6 CONCLUSIONS AND FUTURE WORK

In this the project the process of technology selection for the construction of a new software for the management of digital Archives for the ANTT was documented. Interoperability is a main concern of this project—thus, the adoption of CIDOC CRM is considered a non-functional requirement, since it enables compliance with the ISO 21127 standard.

It was concluded that a relational database would not be capable of dealing with all possible types of entities and relationships in the model, since it is a rich ontology. As such, the objective of the project became the creation of a transactional system based on a graph database. This choice was made due to the sophistication of the CIDOC CRM model, since it uses multiple inheritance, as well as domains, ranges and cardinalities in the properties that it defines. This creates challenges to the programming of software solutions.

Given the weight of the consistency validations in a graph model when compared with a relational model, it is more common to see graph databases and triple stores in OLAP/WORM applications instead of OLTP solutions. Thus, it is a challenge to ensure the availability that one expects of an OLTP system while at the same time ensuring compliance with the CIDOC CRM.

In order to make it easier to program the system, we searched for a set of libraries that could be used to enforce the ontological consistency of the database. Few articles cover the validation of consistency and it is a subject that has been only briefly explored as a graph database benchmark task, unlike synthetic benchmarking, which is more common.

To evaluate both the performance and the consistency validation capabilities offered by each abstraction layer considered, we created an experiment that attempted to represent a relatively simple graph, containing test scenarios for the requirements considered essential to the correct mapping of the CRM in a software solution. These are multiple inheritance (which handles the Diamond Problem) as well as the validation of domain, range and cardinality of properties.

After implementing the model using each of the technology stacks, we concluded that the utilization of the neomodel OGM with a Neo4j database is the only solution that satisfies the proposed requirements. However, a second benchmark test that consists in the creation and deletion of the model multiple times, concludes that this stack is the slowest among all analyzed alternatives.

A common trait among all solutions is the lack of maturity, which lead us to adopt the solution more relatively mature and officially endorsed by the creators of the database, even if it is not the fastest. No results were obtained for the GVerse + DGraph stack, as it was not possible to configure the GVerse OGM successfully.

The EPISA project will explore knowledge graph expansion in the future. Thus, we plan to implement an ETL pipeline which will take data from the OLTP system and enrich it by linking related resources from

Table 2. Comparative overview of OGM

OGM/Dimension	GVerse	OrangoJS	ArangoJs	Neomodel
Graph Database	DGraph	ArangoDB	ArangoDB	Neo4j
Properties	ACID [22]	ACID [16]	ACID [16]	ACID [16]
Programming Language	Typescript	JavaScript	JavaScript	Python
Supports multiple inheritance	?	✖	✔	✔
Eiffel Tower model	?	✖	✖	✔
1000 insertions of the model duration average (s)	NA	9.6	61.3	78.3
10000 insertions of the model duration average (s)	NA	116.4	591.4	957.5

existing knowledge bases such as dbpedia or wikidata. In terms of programming, the OGM could make the ETL module easier to maintain, since it provides an Object-Oriented alternative to scripted Cypher queries.

ACKNOWLEDGMENTS

This work is financed by National Funds through FCT - Foundation for Science and Technology I.P., within the scope of the EPISA project - DSAIPA/DS/0023/2018.

REFERENCES

[1] Nayak Ameya, Poriya Anil, and Poojary Dikshay. 2013. Type of NOSQL databases and its comparison with relational databases. *International Journal of Applied Information Systems* 5, January 2013 (2013), 16–19.

[2] Renzo Angles. 2012. A comparison of current graph database models. In *Proceedings - 2012 IEEE 28th International Conference on Data Engineering Workshops, ICDEW 2012*. IEEE, 171–177. https://doi.org/10.1109/ICDEW.2012.31

[3] ArangoDB. 2020. *ArangoDB*. ArangoDB. https://www.arangodb.com/

[4] Sören Auer and Heinrich Herre. 2007. A Versioning and Evolution Framework for RDF Knowledge Bases. In *Perspectives of Systems Informatics*, Irina Virbitskaite and Andrei Voronkov (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 55–69.

[5] Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The semantic web. *Scientific american* 284, 5 (2001), 34–43.

[6] Jeen Broekstra and Arjohn Kampman. 2003. Inferencing and Truth Maintenance in RDF Schema. In *PSSS1 - Practical and Scalable Semantic Systems, Proceedings of the First International Workshop on Practical and Scalable Semantic Systems, Sanibel Island, Florida, USA, October 20, 2003 (CEUR Workshop Proceedings, Vol. 89)*, Raphael Volz, Stefan Decker, and Isabel F. Cruz (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-89/broekstra-et-al.pdf

[7] Ozgu Can, Emine Sezer, Okan Bursa, and Murat Osman Unalir. 2017. Comparing relational and ontological triple stores in healthcare domain. *Entropy* 19, 1 (2017), 30.

[8] Steve Cassidy and James Ballantine. 2007. Version control for RDF triple stores. *ICSOFT 2007 - 2nd International Conference on Software and Data Technologies, Proceedings* ISDM, WSEHS/- (2007), 5–12.

[9] World Wide Web Consortium. 2017. *Shapes Constraint Language (SHACL)*. World Wide Web Consortium. https://www.w3.org/TR/shacl/

[10] World Wide Web Consortium. 2020. *Large Triple Stores*. World Wide Web Consortium. https://www.w3.org/wiki/LargeTripleStores

[11] Felix Dietze, Johannes Karoff, André Calero Valdez, Martina Ziefle, Christoph Greven, and Ulrik Schroeder. 2016. An Open-Source Object-Graph-Mapping Framework for Neo4j and Scala: Renesca. In *Availability, Reliability, and Security in Information Systems*, Francesco Buccafurri, Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl (Eds.). Springer International Publishing, Cham, 204–218.

[12] Felix Dietze, Johannes Karoff, André Calero Valdez, Martina Ziefle, Christoph Greven, and Ulrik Schroeder. 2016. An Open-Source Object-Graph-Mapping Framework for Neo4j and Scala: Renesca. In *Availability, Reliability, and Security in Information Systems - IFIP WG 8.4, 8.9, TC 5 International Cross-Domain Conference, CD-ARES 2016, and Workshop on Privacy Aware Machine Learning for Health Data Science, PAML 2016, Salzburg, Austria, August 31 - September 2, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9817)*, Francesco Buccafurri, Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar R. Weippl (Eds.). Springer, 204–218. https://doi.org/10.1007/978-3-319-45507-5_14

[13] David Dominguez-Sal, Norbert Martinez-Bazan, Victor Muntes-Mulero, Pere Baleta, and Josep Lluis Larriba-Pey. 2011. A Discussion on the Design of Graph Database Benchmarks. In *Performance Evaluation, Measurement and Characterization of Complex Systems*, Raghunath Nambiar and Meikel Poess (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 25–40.

[14] Roland Ducournau and Jean Privat. 2011. Metamodeling semantics of multiple inheritance. *Science of Computer Programming* 76, 7 (2011), 555–586.

[15] Diogo Fernandes and Jorge Bernardino. 2018. Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. In *Proceedings of the 7th International Conference on Data Science, Technology and Applications, DATA 2018, Porto, Portugal, July 26-28, 2018*, Jorge Bernardino and Christoph Quix (Eds.). SciTePress, 373–380. https://doi.org/10.5220/0006910203730380

[16] Diogo Fernandes and Jorge Bernardino. 2018. Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB. In *Proceedings of the 7th International Conference on Data Science, Technology and Applications* (Porto, Portugal) *(DATA 2018)*. SCITEPRESS - Science and Technology Publications, Lda, Setubal, PRT, 373–380. https://doi.org/10.5220/0006910203730380

[17] Peter M. Fischer, Georg Lausen, Alexander Schätzle, and Michael Schmidt. 2015. RDF constraint checking. *CEUR Workshop Proceedings* 1330 (2015), 205–212.

[18] David Grove, Greg DeFouw, Jeffrey Dean, and Craig Chambers. 1997. Call graph construction in object-oriented languages. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA* 32, 10 (1997), 108–124. https://doi.org/10.1145/263700.264352

[19] Matthew Horridge and Sean Bechhofer. 2011. The OWL API: A Java API for OWL ontologies. *Semantic Web* 2, 1 (2011), 11–21. https://doi.org/10.3233/SW-2011-0025

[20] Jiewen Huang, Daniel J. Abadi, and Kun Ren. 2011. Scalable SPARQL Querying of Large RDF Graphs. *Proc. VLDB Endow.* 4, 11 (Aug. 2011), 1123–1134. https://doi.org/10.14778/3402707.3402747

[21] Eero Hyvönen, Erkki Heino, Petri Leskinen, Esko Ikkala, Mikko Koho, Minna Tamper, Jouni Tuominen, and Eetu Mäkelä. 2016. WarSampo data service and semantic portal for publishing linked open data about the second world war history. In *European Semantic Web Conference*. Springer, 758–773.

[22] Manish Jain and Dgraph Labs. 2020. Dgraph: Synchronously Replicated, Transactional and Distributed Graph Database. (2020).

[23] Nishtha Jatana, Sahil Puri, Mehak Ahuja, Ishita Kathuria, and Dishant Gosain. 2012. A survey and comparison of relational and non-relational database. *International Journal of Engineering Research & Technology* 1, 6 (2012), 1–5.

[24] Salim Jouili and Valentin Vansteenberghe. 2013. An Empirical Comparison of Graph Databases. In *International Conference on Social Computing, SocialCom 2013, SocialCom/PASSAT/BigData/EconCom/BioMedCom 2013, Washington, DC, USA, 8-14 September, 2013*. IEEE Computer Society, 708–715. https://doi.org/10.1109/SocialCom.2013.106

[25] Inês Koch, Nuno Freitas, Cristina Ribeiro, Carla Teixeira Lopes, and João Rocha da Silva. 2019. Knowledge Graph Implementation of Archival Descriptions Through CIDOC-CRM. In *Digital Libraries for Open Knowledge*, Antoine Doucet, Antoine Isaac, Koraljka Golub, Trond Aalberg, and Adam Jatowt (Eds.). Springer International Publishing, Cham, 99–106. https://doi.org/10.1007/978-3-030-30760-8_8

[26] Inês Koch, Cristina Ribeiro, and Carla Teixeira Lopes. 2020. ArchOnto, a CIDOC-CRM-Based Linked Data Model for the Portuguese Archives. In *Proceedings of the 24th International Conference on Theory and Practice of Digital Libraries*, Mark Hall, Tanja Merčun, Thomas Risse, and Fabien Duchateau (Eds.). Springer International Publishing, Cham, 133–146. https://doi.org/10.1007/978-3-030-54956-5_10

[27] Ioannis Konstantinou, Evangelos Angelou, Christina Boumpouka, Dimitrios Tsoumakos, and Nectarios Koziris. 2011. On the elasticity of NoSQL databases over cloud management platforms. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 2385–2388.

[28] Karl-Heinz Lampe, Sigfried Krause, and Martin Doerr. 2010. The CIDOC Conceptual Reference Model (CIDOC-CRM): PRIMER. *CIDOC-CRM official web site* 53 (2010), 333–338. http://www.cidoc-crm.org/

[29] Neal Leavitt. 2010. Will NoSQL databases live up to their promise? *Computer* 43, 2 (2010), 12–14.

[30] Pierre Maillot, Thomas Raimbault, David Genest, and Stéphane Loiseau. 2014. Consistency Evaluation of RDF Data: How Data and Updates are Relevant. In *Tenth International Conference on Signal-Image Technology and Internet-Based Systems, SITIS 2014, Marrakech, Morocco, November 23-27, 2014*, Kokou Yétongnon, Albert Dipanda, and Richard Chbeir (Eds.). IEEE Computer Society, 187–193. https://doi.org/10.1109/SITIS.2014.39

[31] Robert Campbell McColl, David Ediger, Jason Poovey, Dan Campbell, and David A. Bader. 2014. A performance evaluation of open source graph databases. In *Proceedings of the first workshop on Parallel programming for analytics applications, PPAA 2014, Orlando, Florida, USA, February 16, 2014*, Manoj Kumar, Joefon Jann, and Priya Nagpurkar (Eds.). ACM, 11–18. https://doi.org/10.1145/2567634.2567638

[32] Gavin Mendel-Gleason, Kevin Feeney, and Rob Brennan. 2015. Ontology Consistency and Instance Checking for Real World Linked Data. In *Proceedings of the 2nd Workshop on Linked Data Quality co-located with 12th Extended Semantic Web Conference (ESWC 2015), Portorož, Slovenia, June 1, 2015 (CEUR Workshop Proceedings, Vol. 1376)*, Anisa Rula, Amrapali Zaveri, Magnus Knuth, and Dimitris Kontokostas (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-1376/LDQ2015_paper_03.pdf

[33] Justin J Miller. 2013. Graph database applications and concepts with Neo4j. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, Vol. 2324.

[34] A. B. M. Moniruzzaman and Syed Akhter Hossain. 2013. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *CoRR* abs/1307.0191 (2013). arXiv:1307.0191 http://arxiv.org/abs/1307.0191

[35] Neo4j. 2020. *Rdf triple stores vs. labeled property graphs: what's the difference?* Neo4j. https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/

[36] Mohamed A Mohamed Obay, G Altrafi, and Mohammed O Ismail. 2014. Relational vs . NoSQL Databases : A Survey. *International Journal of Computer and Information Technology* 03, 03 (2014), 2279–764.

[37] Damyan Ognyanov and Atanas Kiryakov. 2002. Tracking Changes in RDF(S) Repositories. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, Asunción Gómez-Pérez and V. Richard Benjamins (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 373–378.

[38] Dominic Oldman and Diana Tanase. 2018. Reshaping the Knowledge Graph by Connecting Researchers, Data and Practices in ResearchSpace. In *The Semantic Web – ISWC 2018*, Denny Vrandečić, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl (Eds.). Springer International Publishing, Cham, 325–340.

[39] Orango. 2018. *Orango*. Orango. https://orango.js.org/

[40] Ian Robinson, Jim Webber, and Emil Eifrem. 2013. *Graph databases*. " O'Reilly Media, Inc.".

[41] Yannis Roussakis, Ioannis Chrysakis, Kostas Stefanidis, Giorgos Flouris, and Yannis Stavrakas. 2015. A Flexible Framework for Understanding the Dynamics of Evolving RDF Datasets. In *The Semantic Web - ISWC 2015*, Marcelo Arenas, Oscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, Krishnaprasad Thirunarayan, and Steffen Staab (Eds.). Springer International Publishing, Cham, 495–512.

[42] Philip Russom. 2011. BIG DATA ANALYTICS - TDWI BEST PRACTICES REPORT Introduction to Big Data Analytics. *TDWI best practices report, fourth quarter* 19, 4 (2011), 1–34. https://vivomente.com/wp-content/uploads/2016/04/big-data-analytics-white-paper.pdf

[43] Maribel Yasmina Santos and Isabel Ramos. 2006. *Business Intelligence: tecnologias da informação na gestão de conhecimento*. FCA-Editora de Informática, Lda.

[44] Chandan Sharma and Roopak Sinha. 2019. A Schema-First Formalism for Labeled Property Graph Databases: Enabling Structured Data Loading and Analytics. In *Proceedings of the 6th IEEE/ACM International Conference on Big Data Computing, Applications and Technologies* (Auckland, New Zealand) *(BDCAT '19)*. Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/3365109.3368782

[45] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. 2007. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics* 5, 2 (2007), 51 – 53. https://doi.org/10.1016/j.websem.2007.03.004 Software Engineering and the Semantic Web.

[46] Dmitry Tsarkov and Ian Horrocks. 2006. FaCT++ Description Logic Reasoner: System Description. In *Automated Reasoning*, Ulrich Furbach and Natarajan Shankar (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 292–297.

[47] Rik Van Bruggen. 2014. *Learning Neo4j*. Packt Publishing Ltd.

[48] Muriel Van Ruymbeke, Pierre Hallot, and Roland Billen. 2017. Enhancing CIDOC-CRM and compatible models with the concept of multiple interpretation. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2 (2017), 287–294.

[49] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. 2010. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference, 2010, Oxford, MS, USA, April 15-17, 2010*, H. Conrad Cunningham, Paul Ruth, and Nicholas A. Kraft (Eds.). ACM, 42. https://doi.org/10.1145/1900008.1900067

A ORANGOJS CODE

```
async function modeling_P130() {
    const schema = new orango.Schema({message: String})
```

```
    schema.type('edge', {from: 'E24_physical_human_made_thing',to: ['E70_thing']})
    return orango.model('P130_shows_feature_of', schema)
}
async function modeling_E24() {
    class E24_physical_human_made_thingSchema extends orango.Schema {
        get getName() {return this.name;}
    }
    const schema = new E24_physical_human_made_thingSchema({name: {type: String,required: true
    },}, {strict: true})
    schema.addIndex(SCHEMA.INDEX.HASH, 'name')
    const E24_physical_human_made_thing = orango.model('E24_physical_human_made_thing', schema)
}
async function modeling_E70() {
    class E70_thingSchema extends orango.Schema {
        get getName() {return this.name;}
    }
    const schema = new E70_thingSchema({
        name: {type: String,required: true},}, {strict: true})
    schema.addIndex(SCHEMA.INDEX.HASH, 'name')
    const E70_thing = orango.model('E70_thing', schema)
}
const E24_physical_human_made_thing = orango.model('E24_physical_human_made_thing')
await E24_physical_human_made_thing.import([{_key: '1', name: 'Tokyo Tower'}])

const E70_thing = orango.model('E70_thing')
await E70_thing.import([{_key: '1',name: 'Eiffel Tower'}])

const P130_shows_feature_of = orango.model('P130_shows_feature_of')
await P130_shows_feature_of.import([{_from: 'e24_physical_human_made_things/1',
    _to: 'e70_things/1', message: 'P130_shows_feature_of edge'}])
```

Listing 1. Modeling the excerpt of the data model with OrangoJs [22]

B ARANGOJS CODE

```
class E24_Physical_Human_Made_Thing {
    constructor(graph1, collectionName) {this.graph1 = graph1;
    this.collectionName = collectionName}

    async saveNode(name) {
        const collections = await this.graph1.listVertexCollections();
        const E24 = this.graph1.vertexCollection(this.collectionName);
        if (!collections.includes(E24.name))
            await this.graph1.addVertexCollection(this.collectionName);
        return await E24.save({name: name});
    }
}
class E70_Thing {
    constructor(graph1, collectionName) {this.graph1 = graph1;
        this.collectionName = collectionName}

    async saveNode(name) {
        const collections = await this.graph1.listVertexCollections();
```

[22] https://github.com/feup-infolab/Comparison_OGMs/tree/master/orangoJs

```
        const E70 = this.graph1.vertexCollection(this.collectionName);
        if (!collections.includes(E70.name))
            await this.graph1.addVertexCollection(this.collectionName);
        return await E70.save({name: name});
    }
}
let P130 = {async p130(e24, e70) {
        const edge_collections = await this.graph1.listEdgeCollections();
        if (!edge_collections.includes('p130-shows-features-of')) {
            await this.graph1.addEdgeDefinition({collection: 'p130-shows-features-of', from: ['
    e24-physical-human-made-thing'],to: ['e70-thing']});}
        const collection = this.graph1.edgeCollection("p130-shows-features-of");
        const edge = await collection.save({ some: "data3" },e24._id, e70._id );
    }
};
const E70_obj = new E70_Thing(graph, 'e70-thing');
const E24_obj = new E24_Physical_Human_Made_Thing(graph, 'e24-physical-human-made-thing');
const E24_doc = await E24_obj.saveNode("Tokyo Tower");
const E70_doc = await E70_obj.saveNode("Eiffel Tower");
await E24Col.p130(E24_doc, E70_doc);
```

Listing 2. Modeling the excerpt of the data model with ArangoJs[23]

C NEOMODEL CODE

```
class E1_CRM_Entity(StructuredNode):
    name = StringProperty(unique_index=True, required=True)
    uid = UniqueIdProperty()
    P1_is_identified_by = RelationshipTo("E42_Identifier","P1_is_identified_by")
    P48_has_preferred_identifier = RelationshipTo("E42_Identifier", "
    P48_has_preferred_identifier", cardinality=ZeroOrOne)
class E70_Thing(E1_CRM_Entity):
    name = StringProperty(unique_index=True, required=True)
    uid = UniqueIdProperty()
    P130_shows_features_of = RelationshipTo("E70_Thing","P130_shows_features_of")
class E71_Man_Made_Thing(E70_Thing):
    name = StringProperty(unique_index=True, required=True)
    uid = UniqueIdProperty()
    P102_has_title = RelationshipTo("E35_Title","P102_has_title")
class E18_Physical_Thing(E70_Thing):
    name = StringProperty(unique_index=True, required=True)
    uid = UniqueIdProperty()
    P156_occupies = RelationshipTo("E53_place", "P156_occupies")
class E24_Physical_Human_Made_Thing(E18_Physical_Thing, E71_Man_Made_Thing):
    name = StringProperty(unique_index=True, required=True)
    uid = UniqueIdProperty()
e24 = E24_Physical_Human_Made_Thing(name="Tokyo Tower").save()
e70 = E70_Thing(name="Eiffel Tower").save()
e24.P130_shows_features_of.connect(e70)
```

Listing 3. Modeling the excerpt of the data model with Neomodel [24]

[23] https://github.com/feup-infolab/Comparison_OGMs/tree/master/ArangoJS/arangojstest
[24] https://github.com/feup-infolab/Comparison_OGMs/tree/master/Neomodel