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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2021 Association for Computing Machinery.
Manuscript submitted to ACM

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 classi cation previously mentioned there are also other classi cations 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 o ers 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] o ers 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 di erent 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 modi cations are performed, versioning should be available for audit purposes, and nally, 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 o er 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 o er better e ciency 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 quali ed 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 de ned 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-speci c 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 exibility 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 e ciently 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 modi cation 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-de ned 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.