TyphonML: a Modeling Environment to Develop Hybrid Polystores

Francesco Basciani, Juri Di Rocco, Davide Di Ruscio, Alfonso Pierantonio
University of L'Aquila
L'Aquila, Italy
{firstname.lastname}@univaq.it

Ludovico Iovino
Gran Sasso Science Institute
L'Aquila, Italy
ludovico.iovino@gssi.it

ABSTRACT

Designing and deploying a hybrid data persistence architecture that involves a combination of relational and NoSQL databases is a complex, technically challenging, and error-prone task. In this tool paper, we propose *TyphonML*, a modeling language and supporting environment, which permits modelers to specify data that need to be persisted in hybrid architectures, by abstracting over the specificities of the underlying technologies. The language enables the specification of both conceptual entities and available data layer technologies, and then how the modeled entities have to be mapped to the available database systems. *TyphonML* models are used to generate microservice-based infrastructures, which permit users to interact with the designed hybrid polystores at the conceptual level. In this tool paper, we show the different components of the *TyphonML* environment at work through a demonstration scenario.

CCS CONCEPTS

• **Software and its engineering → Model-driven software engineering**; **Software system models**; **Software system structures**; *Software organization and properties.*

KEYWORDS

Hybrid Polystore, Data Modelling, Tools, Database Technologies.

ACKNOWLEDGMENTS

This work is funded by the European Union Horizon 2020 research and innovation programme through the Polyglot and Hybrid Persistence Architectures for Big Data Analytics (TYPHON) project (#780251).

1 INTRODUCTION

Relational database management systems (RDBMS) have become over the years the predominant choice for storing large volumes of data. As such, various techniques and tools have been developed to support their design and development. In recent years, NoSQL databases have emerged as an alternative approach to data storage thanks to their horizontal scalability and flexibility, even though NoSQL databases still remain far from the level of maturity of relational databases. To balance requirements for data consistency and availability, organisations increasingly migrate towards *hybrid* data persistence architectures comprising both relational and NoSQL databases for managing different subsets of their data. Designing and deploying such hybrid data stores (hereafter referred as polystores for conciseness) is a complex, technically challenging and error-prone task. Differently from relational databases which can be accessed by means of standard application programming interfaces (APIs) such as JDBC/ODBC and by relying on the support for SQL, each NoSQL technology provides developers with its own proprietary API and query language, with a consequent high coupling between the employed databases and the developed applications.

The TYPHON EU H2020 project[1] aims at conceiving an industry-validated methodology and integrated technology offering for designing, developing, querying, evolving, analysing and monitoring architectures for scalable polystores. In the context of the TYPHON project, the *TyphonML* language has been developed to permit engineers to model in a homogeneous manner polystores, by abstracting over the specificities of the underlying technologies. Both textual and graphical editors have been developed to support the specification of *TyphonML* models. Moreover, ready-to-use APIs are automatically generated from input *TyphonML* specifications so that developers are able to perform CRUD (create, read, update and delete) operations on hybrid polystores in a homogeneous manner without the need of directly interacting with the low level infrastructures. To this end a microservice architecture is generated enriched with an OpenAPI[2] specification allowing both humans and machines to transparently interact with the polystore. *TyphonML* has been developed by exploiting mature Eclipse technologies, including EMF, Sirius, XText, OCL, and Acceleo.

Structure of the Paper: Section 2 motivates the work and makes and overview of the related background. Section 3 presents the *TyphonML* language and the supporting development environment. Conclusion and future work are presented in Section 4.

A video demonstration of the *TyphonML* tool is available at https://bit.ly/typhonml-models-tools – The source code of the tool is available at https://github.com/typhon-project/typhonml

2 BACKGROUND AND MOTIVATION

Figure 1 shows a motivating e-commerce application borrowed from [10]. The depicted system relies on different kinds of database systems: a graph database is employed to support product analysis, document database for managing user reviews and comments, relational database for orders and payments, products, and finally a key-value technology for storing product images. A search engine is also available for managing products, reviews and comments.

Figure 1 shows an explanatory case based on the adoption of a hybrid data persistence architecture for managing the data of the

[1] https://www.typhon-project.org/
[2] https://www.openapis.org/

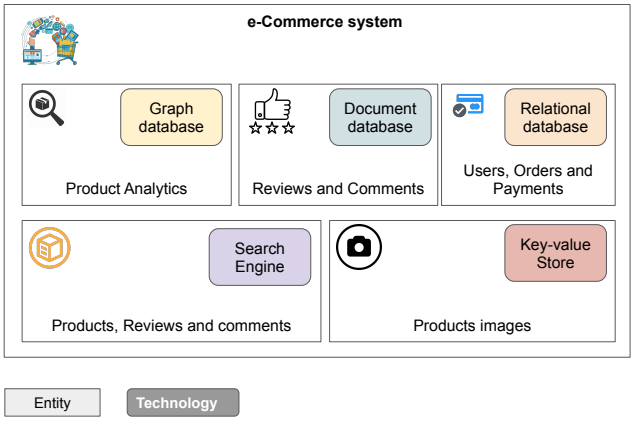


Figure 1: eCommerce running example

same software system. Designing and implementing such architectures can be technically challenging and error-prone. For instance, standardized notations and supporting tools for designing NoSQL databases are not available. While there is some work towards this direction, the proposed solutions are technology-specific and not applicable across different and hybrid datasets [1].

Over the last years, several approaches have been proposed to support the development of systems similar to that shown in Fig. 1. In [4, 7], the authors present BigDAWG, a polystore system designed to work with different data models and storage engines grouped in islands. It includes a middleware to support the multiple islands, programming languages and query types that BigDAWG supports. Moreover, BigDAWG provides an API interface to execute polystore queries. AWESOME [3] is a polystore-based system to support social data analytics. Social data coming from different sources (e.g., DBPedia, Twitter, etc) are ingested from the polystore to enable data analytics. In [12], the authors propose a methodological approach to analyze the requirements of data warehouse systems. In [5] EI Mohajir et al. present a framework to design a data warehouse project by concurrently taking into account both functional and the non-functional requirements. ORESTES is a tool presented in [8] to provide REST/HTTP access to object-oriented databases. In [14], Digree has been proposed as a middleware to handle graph pattern matching queries over distributed or inter-linked graph databases. It orchestrates the query decomposition into a set of smaller patterns that are executed in parallel over all database partitions.

Even though several techniques and tools have been done in the direction of supporting the design and development of hybrid storage systems, a satisfactory solution is still missing: the existing approaches do not focus on the separation of conceptual and storage concepts. Moreover, existing tools do not provide proper abstractions from technological aspects and in most of the cases a centralized data layer for accessing data among different technologies is still not supported.

In the next section, we propose *TyphonML*, a tool-supported modeling language to specify conceptual entities and map them to different storage engines. Then, a microservice-based data layer is automatically generated to permit the interaction with the polystore in a homogenous and transparent manner from the adopted database technologies. Developers interact with the polystore at the conceptual level, without the need of being aware of how the data entities of interest are stored.

2.1 Typhon H2020 EU project

TyphonML is a component of the Typhon EU project. This project aims at conceiving an integrated technology offering for designing, developing, querying, evolving, analysing and monitoring architectures for scalable polystores. In the following we briefly present the other components of Typhon project:

- The *TyphonDL* component provides a language for modeling hybrid polystore deployments. It enhances the *TyphonML* models with deployment settings (e.g., security settings, docker configurations, etc.). Then, it generates installation and configuration scripts to deploy the persistence infrastructure.
- The *TyphonQL* component provides a unified language for querying data in polystores. It includes many features as the compilation to native querying facilities, the static analysis, and the lazy loading and prefetching mechanisms.
- The *Typhon Monitoring and Analytics* component records data access, update events and data events in a distributed ledger. Then, it uses these data to facilitate subscription-based data analytics and text mining pipelines.
- The *Typhon Evolution* component supports schema evolutions, internal data migrations, and query migration tools. Moreover, it uses usage monitoring data to recommend possible evolution schemata.

3 THE TYPHONML ENVIRONMENT

The architecture of the developed environment is presented in Section 3.1. Both the *TyphonML* textual and graphical editors are presented in Section 3.2. To support the specification of *TyphonML* models and provide the modelers with early feedback, specific validators and recommenders are available as discussed in Section 3.3. The generation of the data access layer based on a micro-service architecture is presented in Section 3.4.

3.1 Architecture

The overall architecture of the *TyphonML* environment is shown in Fig. 2: white boxes (i.e., EMF, Xtext, Epsilon, Sirius, and Acceleo) represent the technologies that have been used; dark-grey boxes depict the software components the modeler interact directly with; light-grey boxes present inner software components that support the functionalities provided to the user via the editors and the data access layer.

The *TyphonML* metamodel (labeled with ① in Fig. 2) plays a key role in the overall architecture. It contains the modeling constructs enabling the specification of the data types and conceptual entities of the information system being modeled. Once modeled, data entities can be then mapped to specific databases also modeled by means of the same language. A fragment of the the *TyphonML* metamodel is shown in Fig. 3. A *TyphonML* model mainly consists of `DataType` and `Database` elements. `DataType` instances represent all the (primitive and compound) data types that are used by the system under development. A particular kind of datatype is represented by `Entity` that permits to specify the conceptual entities involved

in the information system under development. Each entity is in turn defined by means of structural features, i.e., `Attributes` and `Relations`. `Attribute` elements can typed as `primitive`, `custom`, or even as a modeled `Entity`. `Relations` are named elements representing relationships among different entities including the corresponding `cardinality`, which can be `zero_one`, `one`, `zero_many`, or `one_many`. Bidirectional references can be specified by using the `opposite` reference between the `Relation` instances. Finally, `isContainment` is a boolean attribute, which permits to specify if the target entity is contained (e.g., to trigger cascade-deletions) or not in the entity being modeled.

`Database` is an abstract concept, which is specialized in order to specifically represent different kinds of database systems. The currently supported subtypes are `RelationalDB`, `DocumentDB`, `KeyValueDB`, `ColumnDB`, and `GraphDB`. For the sake of brevity, only `RelationalDB`, and `DocumentDB` are detailed in the following.

`RelationalDB` allows to define which entity should be stored in a relational database, instantiating a mapping. To this end, modelers will define the `Tables` that are needed to store the entity of interests. `Table` contains the reference to the `entity` that needs to be stored in the relational database being specified. Similarly, document based databases are used to store collections of heterogeneous elements. The `DocumentDB` metaclass consists of the reference `collections` to group documents that are stored in the database. Each `Collection` is devoted to store the data of the referred conceptual data `entity`. The metamodel presented above has been implemented as an `Ecore` model on top of the EMF platform [2].

3.2 Textual and Graphical Editors

The *TyphonML* textual editor has been developed by means of Xtext[3], which is an Eclipse project for developing domain-specific languages. Starting from the specification of the grammar (see ② in Fig. 2), the Xtext framework supports the implementation of a full infrastructure, including parser, linker, type-checker, compiler as well as editing support for Eclipse. Figure 4(a) shows the textual specification of a *TyphonML* model representing the explanatory e-commerce system introduced in Section 2. For instance, the conceptual entity `User` specified on the left-hand side of Fig. 4(a) is

[3]https://www.eclipse.org/Xtext/

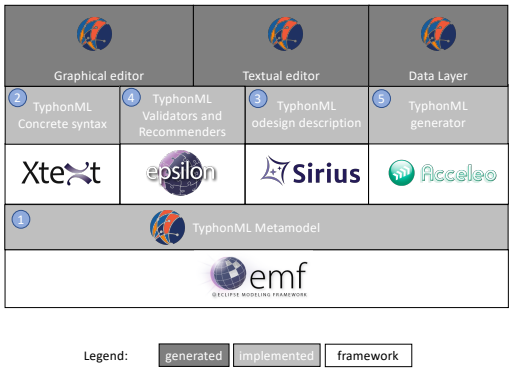


Figure 2: General View of the presented tool

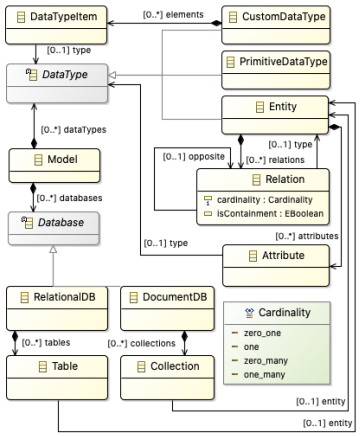


Figure 3: An excerpt of the *TyphonML* metamodel

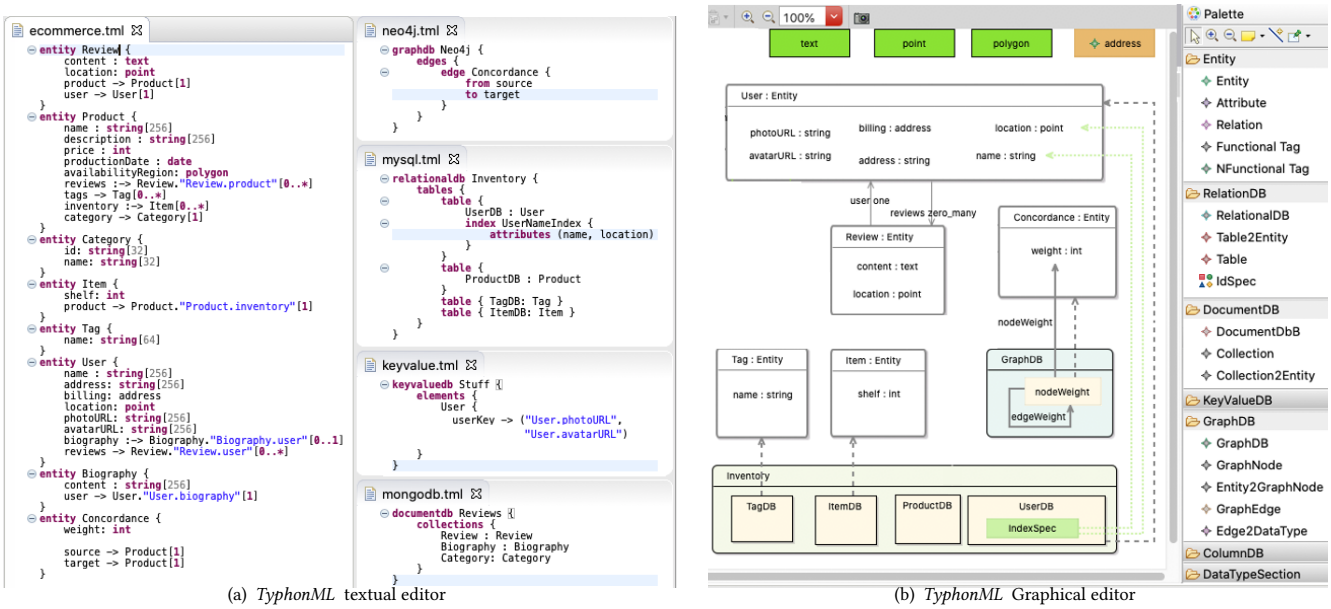
mapped on the table `UserDB` of the relational database `Inventory`. The other conceptual entities have been also mapped to the different available database types, e.g., the `Review` entity is mapped to a document database named `Reviews`. Such data mappings are made more explicit in the graphical editor shown in Figure 4(b).

The *TyphonML* graphical editor has been developed by means of Sirius[4]. Sirius is an Eclipse project that enables the development of graphical modeling environments by leveraging well-established technologies. Starting from a metamodel, it allows a model-based specification of visual concrete syntax organized in viewpoints (pointed with ③), i.e., models that can be authored by means of different notations that suit the needs of various stakeholders. Thanks to the Sirius and Xtext integration, graphical and textual editors are both synchronized. In this way, stakeholders with different skills can define *TyphonML* models using different syntaxes [6].

3.3 TyphonML validators and recommenders

In database systems, the possible violation of recommended best practices might potentially affect negatively the quality of the system being developed. Thus, early detection of potential issues while specifying database schemas could help developers enhance quality aspects of the model system [13]. In *TyphonML* a quality assurance checker has been integrated, in which models are analyzed by a set of checks, each devoted to the discovery of possible issues. The conceived analysis tools have been developed by relying on Epsilon Object Language (EOL)[5] [9] and Epsilon Validation Language (EVL)[6] provided by the Epsilon platform. EOL is an imperative programming language for creating, querying, and modifying EMF models, while EVL is a validation language built on top of EOL providing a number of features such as support for detailed user feedback, constraint dependency management, semi-automatic transactional inconsistency resolution. According to the work presented in [13], we defined a set of model validators (④ in Figure 2) working at *Schema*, and *Data* levels. An example of issue occurring

[4]https://eclipse.org/sirius/
[5]https://www.eclipse.org/epsilon/doc/eol/
[6]https://www.eclipse.org/epsilon/doc/evl/



(a) *TyphonML* textual editor (b) *TyphonML* Graphical editor

Figure 4: *TyphonML* editors.

at the schema level is when containment relations are specified between conceptual entities that are mapped to relational databases. Even though this is technically possible, containments in relational databases should be avoided, and for performance purposes this kind of relations should be managed by means of NoSQL databases e.g., document databases. An example of issue occurring at data level is when 'O' (letter) is used instead of '0' (zero number). When potential issues are identified, possible fixes are recommended directly in the TyphonML environment and the modeler can decide to accept them and thus change the model accordingly.

3.4 Generation of the Data Access Layer

Once the *TyphonML* specification is completed and validated as previously discussed, a synthesis tool is applied to generate the polystore data access layer. A client applications can use it as an intermediate layer to transparently interact with the underlying database systems instead of directly working with each of them. For instance, by considering the *TyphonML* model of the explanatory example shown in Fig. 4, the data access layer shown in Fig. 5 is generated. In particular, each modeled database system induces the creation of a corresponding microservice [11], which is responsible of managing all the conceptual entities that have been assigned to that database system. The architecture consists of a *Client Library* that can be used by the developers to interact with the interfaces exposed by the *API Gateway*. The *API Gateway* is the service aware of "where" all the services managing the conceptual entities are deployed, and thus it is aware of where the client requests have to be dispatched. The system is also able to manage relationships occurring among entities stored in different database systems, and thus managed by different microservices. For instance, as shown in Fig. 5, the `Product` entity is mapped to a dedicated microservice managing the data with a graph database, i.e., Neo4J, the `User` entity

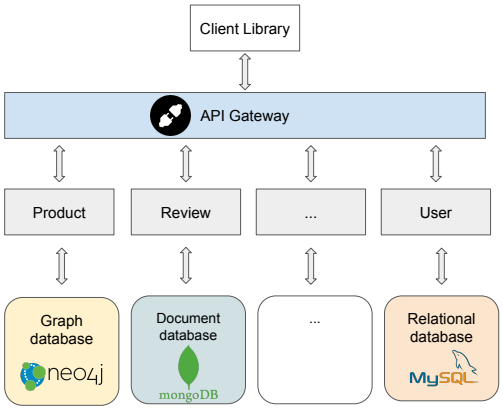


Figure 5: Example of generated microservice-based data access layer

instead is instead managed by a relation database as specified in the model in Figure4. The generation of the data access layer from an input *TyphonML* model, is managed by a set of coordinated Acceleo-based[7] model-to-code transformations (see ⑤ in Figure 2). The generated microservice architecture comes with an automatically produced OpenAPI specification for describing, consuming, and visualizing the corresponding RESTful web services. It allows both humans and machines to interact with the polystore. Figure 6 shows the generated OpenAPI specification of the explanatory example shown by means of Swagger UI[8]. The generated API consists of all the resources and operations that can be invoked, i.e., the CRUD operations GET, POST, DELETE, and PATCH.

[7] https://www.eclipse.org/acceleo/
[8] https://swagger.io/tools/swagger-ui/

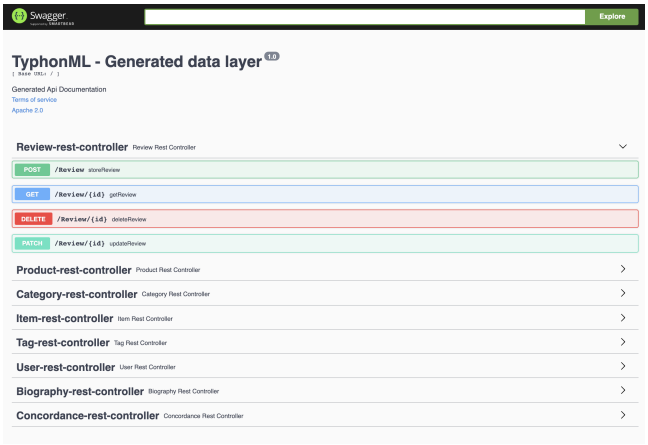


Figure 6: Generated microservice-based architecture for the Polystore API

The generated data access layer, including the RESTful web services, can be programmatically used as shown in Listing 1, which defines an explanatory method `testReviewFindAll` that uses the *Client Library* to retrieve and aggregate data coming from different storage sources. By referring to the motivation example, the entity `Review` is mapped on a document database. Moreover such an entity has two references, i.e., `Product` and `User`, that are both mapped to a relational database. The provided service `ReviewService` interacts with the *API Gateway* to aggregate data coming from different sources. Line 2 instantiates the service `reviewService` objects with the URL of the *API Gateway*, the line 3 performs the query on the polystore. Finally, the contained objects, such as `product`, are deferred the initialization until they are really needed in line 6.

```
1 public void testReviewFindAll() {
2   ReviewService reviewService = new ReviewService("http://localhost:8080");
3   PagedResources<Review> reviews = reviewService.findAll(1,5,"ASC");
4   reviews.forEach( review ->
5     review.getProductObj().forEach(product ->
6       System.out.println(product.getName() + " : " + product.getDescription()))
7     );
8 }
```

Listing 1: An explanatory use of the generated data access layer

4 CONCLUSIONS

Designing and deploying hybrid data persistence architectures that involve combination of different databases is a complex, and error-prone task. In this paper we proposed *TyphonML* , a technical infrastructure for designing hybrid polystores and interact with them in a homogenous manner. *TyphonML* allows the design of conceptual entities and map them to different database systems by means of a textual and a graphical environment. *TyphonML* permits developers to interact with the modeled data at the conceptual level by relying on a data access layer, which is based on a generated microservice-based architecture. Future developments include the support for evolution and migration scenarios. In particular, starting from an existing and already deployed *TyphonML* model, modelers might have the need of operating conceptual and logical changes to the managed data. To this end, we are planning to add a predefined set of *change operators* supported by an *evolution editing mode* of the editors.

REFERENCES

[1] Paolo Atzeni, Francesca Bugiotti, and Luca Rossi. 2014. Uniform access to NoSQL systems. *Information Systems* 43 (2014), 117 – 133. https://doi.org/10.1016/j.is.2013.05.002

[2] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T.J. Grose. 2003. *Eclipse Modeling Framework*. Addison Wesley.

[3] Subhasis Dasgupta, Kevin Coakley, and Amarnath Gupta. 2016. Analytics-driven data ingestion and derivation in the AWESOME polystore. In *2016 IEEE International Conference on Big Data (Big Data)*. 2555–2564. https://doi.org/10.1109/BigData.2016.7840897

[4] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. 2015. The BigDAWG Polystore System. *ACM SIGMOD Record* 44, 2 (Aug. 2015), 11–16. https://doi.org/10.1145/2814710.2814713

[5] Mohammed El Mohajir and Amal Latrache. 2012. Unifying and incorporating functional and non functional requirements in datawarehouse conceptual design. In *2012 Colloquium in Information Science and Technology*. 49–57. https://doi.org/10.1109/CIST.2012.6388062 ISSN: 2327-1884.

[6] Luc Engelen and Mark van den Brand. 2010. Integrating textual and graphical modelling languages. *Electronic Notes in Theoretical Computer Science* 253, 7 (2010), 105–120.

[7] Vijay Gadepally, Peinan Chen, Jennie Duggan, Aaron Elmore, Brandon Haynes, Jeremy Kepner, Samuel Madden, Tim Mattson, and Michael Stonebraker. 2016. The BigDAWG polystore system and architecture. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–6. https://doi.org/10.1109/HPEC.2016.7761636

[8] Felix Gessert, Florian Bücklers, and Norbert Ritter. 2014. Orestes: A scalable Database-as-a-Service architecture for low latency. In *2014 IEEE 30th International Conference on Data Engineering Workshops*. 215–222. https://doi.org/10.1109/ICDEW.2014.6818329 ISSN: null.

[9] Dimitrios S Kolovos, Richard F Paige, and Fiona AC Polack. 2006. The epsilon object language (EOL). In *European Conference on Model Driven Architecture-Foundations and Applications*. Springer, 128–142.

[10] Jiaheng Lu, Irena Holubová, and Bogdan Cautis. 2018. Multi-model databases and tightly integrated polystores: Current practices, comparisons, and open challenges. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 2301–2302.

[11] Sam Newman. 2015. *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.".

[12] Fá Rilston Silva Paim and Jaelson Castro. 2002. Enhancing Data Warehouse Design with the NFR Framework. In *Anais do WER02 - Workshop em Engenharia de Requisitos, Valencia, España, Novembro 11-12, 2002*, Oscar Pastor and Juan Sánchez Díaz (Eds.). 40–57. http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER02/paim.pdf

[13] Tushar Sharma, Marios Fragkoulis, Stamatia Rizou, Magiel Bruntink, and Diomidis Spinellis. 2018. Smelly relations: measuring and understanding database schema quality. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. ACM, 55–64.

[14] Vasilis Spyropoulos, Christina Vasilakopoulou, and Yannis Kotidis. 2016. Digree: A middleware for a graph databases polystore. In *2016 IEEE International Conference on Big Data (Big Data)*. 2580–2589. https://doi.org/10.1109/BigData.2016.7840900