Querying Digital Twin Models

Emilio Carrión[1,2][a], Pedro Valderas[2][b] and Óscar Pastor[2][c]
[1]*Mercadona Tech, Mercadona, Spain*
[2]*PROS – VRAIN, de València, Spain*
emcaryp@gmail.com

Keywords: Digital Twin, Modelling, Query.

Abstract: Digital twins (DTs) have become increasingly complex as they integrate data from diverse heterogeneous sources while requiring strict security controls. This heterogeneity presents challenges to effectively query and aggregate information. The Entity-Relationship Digital Twin (ERDT) model provides a basis for representing both physical entities and their digital counterparts, however it lacks mechanisms to handle system-level queries and ensure secure access. This work extends the ERDT model by introducing query views, high-level abstractions that allows flexible and secure querying of DT data over multiple entities. Furthermore, we validate our approach through a real-world industrial case study within Mercadona's logistics operations where we specifically focus on their truck fleet management system. The results demonstrate that our solution covers data heterogeneity and security constraints while also providing enhanced query capabilities in a production environment.

1 INTRODUCTION

Digital twins (DTs), virtual representations of real-world entities synchronised at a specific frequency and fidelity (Digital Twin Consortium, 2024), have emerged as a revolutionary technology in a wide range of fields, from manufacturing and urban infrastructure to logistics and healthcare (Tao et al., 2019). These virtual counterparts of physical objects and systems enable real-time monitoring, simulation and optimisation, providing advanced insights into operational performance (Jones et al., 2020).

However, as the complexity and scale of DT implementations increase, so does the need for software engineering solutions to ensure quality development. In this context, Model-Driven Development (MDD) seems to be one of the best options (Bordeleau et al., 2020). Currently, there are solutions that address this problem, such as (Kirchhof et al., 2020), (Schroeder et al., 2021) or (Jia et al., 2022), but most of them fail to take into account an important issue for DTs: queries.

One problem when querying DTs is the fact that they often make use of a wide range of data formats and sources to represent their physical counterparts. These sources include 3D models, building information models (BIM), proprietary software systems, physics engines and other domain-specific data formats. This heterogeneity presents a major challenge in querying and aggregating information in a unified way. Without a standardised approach, extracting meaningful information is problematic, as each data source may require specialised management.

In addition, DTs are increasingly used in industries with stringent security requirements. One example is the healthcare field where private personal data must be protected. Ensuring secure access to this data is key, as any unauthorised access could lead to privacy leaks.

Our approach addresses these challenges through a model-based methodology. By abstracting and structuring different types of data within a conceptual modelling framework, we create a standardised query interface that integrates these various data sources and applies security constraints. This approach allows us to address these challenges by defining robust and secure queries in heterogeneous data environments.

In previous research (Carrión et al., 2025), we have introduced The Entity-Relationship Digital Twin (ERDT) model, an extension of the traditional entity-relationship (ER) model. This is a modelling solution that represents both physical entities and their digital counterparts. Although ERDT effectively captures

[a] https://orcid.org/0000-0002-7026-0495
[b] https://orcid.org/0000-0002-4156-0675
[c] https://orcid.org/0000-0002-1320-8471

Carrión, E., Valderas, P. and Pastor, Ó.
Querying Digital Twin Models.
DOI: 10.5220/0013204200003928

In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2025)*, pages 398-405
ISBN: 978-989-758-742-9; ISSN: 2184-4895

the relationships and attributes of entities in a DT, it faces challenges in handling system-level queries and ensuring their access security, as existing ERDT models lack a flexible mechanism for defining and executing queries on multiple entities.

This paper addresses these limitations by introducing a key innovation in the ERDT model: query views. These views provide a high-level abstraction for querying multiple entities by combining and filtering data in a secure and scalable way. These contributions allow for greater flexibility in the way information is accessed and used.

To demonstrate the practical application of these concepts, we also present a case study of a DT of truck fleet management. This real industrial scenario is located within the logistics operations of Mercadona, one of Spain's largest supermarket chains. With more than 1,600 stores and a complex logistics network, Mercadona serves millions of customers daily by ensuring the correct flow of goods from distribution centres to retail locations. The complexity of managing large-scale logistics operations, particularly in the area of truck fleet management, provides an ideal environment to apply and evaluate the effectiveness of DT technologies. By comparing our ERDT-based solution with an existing production environment, we demonstrate its ability to handle data heterogeneity and security constraints in a high-impact, real-world use case.

In summary, this work contributes to improving the state-of-the-art in DT development by:

- Proving a modelling solution based on ERDT views to support flexible query mechanisms that combine data from multiple entities while respecting security constraints.
- Demonstrate the practical application of these innovations through a real industrial case study of a truck fleet management system.
- By improving the query and security capabilities of ERDT models, this work aims to provide a more robust framework for managing the complex data generated by DTs in various domains.

The paper is structured as follows. Section 2 introduces the related work. Section 3 briefly summarises the ERDT model and its limitations in terms of querying and security and presents the real-world case study of a truck fleet management system. Section 4 introduces the core contributions of this work: the use of views on the ERDT model to define queries to DTs and data access policies. Section 5 applies the contribution of this work to a real case study, demonstrating the practical application of views within an ERDT model. Section 6 discusses over the work contributions. Finally, Section 7 provides conclusions and discusses potential future research directions.

2 RELATED WORK

Querying complex models is a well-established concern in the Model-Driven Engineering (MDE) community (Kolovos et al., 2013). Thus, over time, different approaches to address this problem have emerged, in particular in the context of UML and OCL (Object Constraint Language).

Early work (Gogolla and Richters, 1998) used OCL as a query language. These works highlighted the similarity between OCL syntax and traditional database query languages. They also compared OCL and ER model query languages showing that OCL had potential to be used as an expressive query mechanism in object-oriented systems. However, they noted several problems in the definition of OCL that need to be addressed.

Building on this foundation, the work of (Akehurst and Bordbar, 2001) showed that the combination of UML and OCL could achieve the expressiveness of relational algebra, making it ideal for complex query tasks. However, they concluded that specific extensions to OCL were needed for it to be used in practice as a query language, especially when working with UML.

With the emergence of more advanced modelling needs, further advances and improvements in query models have been made. (Stein et al., 2004) proposed a novel modelling notation to represent queries using UML. His proposal introduced dedicated symbols for common selection tasks with OCL semantics. He also stressed the importance of making queries more accessible by visual modelling in a language-agnostic manner.

More recently, models have grown in size and complexity, shifting the focus to making persistence and queries scalable. Morsa, a model repository that uses No-SQL databases to efficiently store models (Espinazo Pagán and García Molina, 2014), is an example of this trend. It shows the need for new approaches to managing large-scale models.

If we talk about DTs, we find even more varied problems. (Bordeleau et al., 2020) highlights that DTs face three major challenges: managing heterogeneous models from different disciplines, maintaining bidirectional synchronization between DTs and physical systems, and supporting collaborative development throughout the entire lifecycle. These challenges directly influence the approach to queries in DT environments.

More recent advances in DT modelling, discussed by (Tao et al., 2022), show the importance of data-driven and model-driven approaches to enable important DT functionalities such as monitoring, simulation, and prediction. Their analysis of the state-of-the-art in DT modelling demonstrates the need for comprehensive modelling solutions that can cover diverse fields and functionalities.

Our work builds on these foundations, but addresses the main challenges posed by DTs. Previous work focused on querying static data, however DTs need real-time access to dynamic data from multiple heterogeneous sources. Our view-based approach extends traditional model query concepts to fit DT-specific needs such as real-time synchronization.

Furthermore, unlike traditional OCL-based approaches that are primarily used as constraint languages, our query views are intended to handle the complex nature of DT data while maintaining security and performance, aligning with (Habela et al., 2008) observations on the limitations of OCL in data processing but providing a solution more tailored to the domain of DTs.

3 ERDT MODEL OVERVIEW

The Entity-Relationship Digital Twin (ERDT) model (Carrión et al., 2025) extends the well-known Entity-Relationship (ER) model (Chen, 1976) to model DTs of physical systems.

The main contributions of this model are the definition of historical values (as DTs often rely on historical data for operational functionalities) and the introduction of interfaces and data flows, key elements in the ERDT model that ensure real-time synchronisation between the physical and virtual components of the DT.

In order to exemplify the use of the ERDT model, we are going to introduce a partial view of a model presented as a case study in a previous work (Carrión et al., 2025). This model represents the last-mile fleet of delivery trucks that we have at Mercadona and its logistics warehouse operations (called hives).

We will use this ERDT model through the work to explain how the model and the proposed extensions are defined and to showcase the relevance of the introduced contributions.

Entity Sets and Relationships. Entities represent the physical objects we are digitising in our DT (see Truck and Driver in Figure 1). Relationships show how entities are related between them.

Entities, as in the base ER model, have attributes. However, our modelling approach introduces support for features characteristic to DTs, such as attributes that need to keep a historical record of their values.

Interfaces. Interfaces act as controlled access points to entities and their attributes, hiding internal complexity and facilitating interaction with the DT (e.g. getLocationHistory() and getMaintenanceDetails()). They are specially useful in systems that manage heterogeneous data sources, as they offer a single access point for fetching or updating information related to an entity, abstracting away the complexity of interacting with multiple systems.

Each entity is equipped with interfaces to handle its queries and updates. This abstraction simplifies interaction and shields users from the complexity of the underlying systems, allowing seamless querying and updates.

Value Sets. Value sets are used in ER models to define a set of values that can be assigned to the attribute of one entity. We use them for the same purpose in the context of ERDT.

Our model contains complex value types that represent information such as GPS coordinate readings or maintenance records.

As we are going to use their definitions in the following sections, we will specify the next details:

- GPSReading[]: an historical value set of coordinate readings. Has an unique() method that returns a new collection with non-repeating coordinate records. It also has a count() method that returns the collection size.
- Polygon (seen in Section 5): a set of values that represents coordinate polygons. Has a contains(coordinate) method that checks if a given coordinate is inside the polygon boundaries.

Data Flows. Data flows represent the bidirectional communications between the DT and the physical components (e.g. Truck GPS Coordinates and Driver Assigned).

In our DT case study, the data flows keep updated the virtual entities with real-time data. Additional information about this aspect can be found in our previous work (Carrión et al., 2025).

In short, interfaces and data flows enable secure and structured information management in the ERDT model. They facilitate bidirectional communication between the DT and the physical components, enabling real-time synchronisation and the ability to

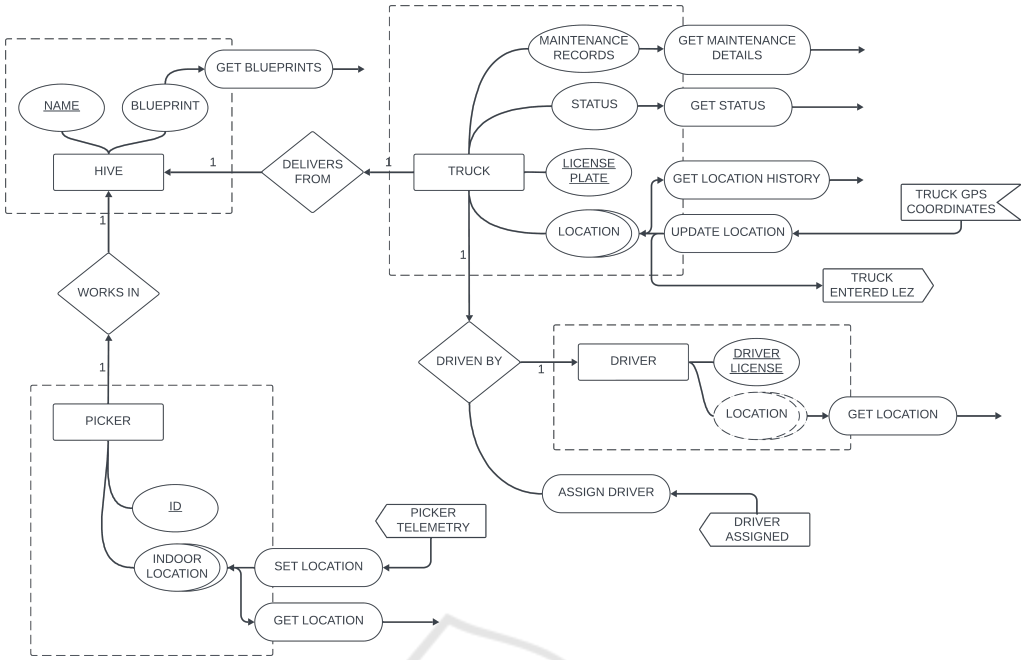

Figure 1: ERDT model of Mercadona's logistics operations.

perform analysis and actions based on the collected data.

3.1 Querying Limitations

Although interfaces in the ERDT model act as a controlled access point to entity attributes and encapsulate internal complexity, they also have some limitations. Mainly, these interfaces work at the entity level, meaning that they are designed to ease interactions with individual entities rather than enabling more complex, system-wide operations that span multiple entities or relationships. This is not useful to define queries or operations that need to aggregate data from different entities in the DT, limiting the flexibility of the model in more complex scenarios.

These limitations show the need for the development of more flexible querying mechanisms, as well as a more comprehensive security model that operates beyond the scope of individual entities.

4 VIEWS AS QUERY ABSTRACTIONS IN ERDT

In the context of DTs, querying data efficiently and securely across multiple entities is essential for gaining insights into complex systems. However, as DTs grow in complexity, relying only on entity-level interfaces can lead to simple, repetitive queries that require users to query each entity's attributes and relationships independently. To address this, we introduce a domain-specific language (DSL) for defining views, which serves as a mechanism to simplify querying and offer higher-level abstractions that enable the reuse of software artifacts (Mernik et al., 2005) in the ERDT model.

Two of the key elements to define a DSL are the abstract syntax and the concrete syntax (Kleppe, 2008). The abstract syntax defines the main concepts of the DSL and their relationships including the rules that define how models can be built. In MDD, this abstract syntax is represented by a formal definition. The concrete syntax provides a notation to represent the abstract syntax.

4.1 Rationale

A view in an ERDT model can be thought of as a pre-defined, system-wide query that aggregates, filters, and transforms data from multiple entities and their relationships. Unlike traditional database views that are static representations of data, ERDT views interact dynamically with entity interfaces executing specific operations across the relevant entities. These views abstract the complexities of querying multiple entities, presenting a coherent result to users without requiring them to interact directly with each entity's interface.

Views are constructed by invoking the operations provided by entity interfaces. Instead of accessing the

low-level data directly views make use of the query operations defined in the interfaces, ensuring that any encapsulation and security constraints present at the entity level are being kept.

A key benefit of views is their ability to combine data from heterogeneous sources and present them in a unified format. This not only simplifies the querying process but also facilitates more complex, system-wide operations.

Views also inherit the security constraints defined at the entity interface level. Because views operate by invoking the operations defined by entity interfaces, they automatically enforce any role-based access control or other security mechanisms that limit which users can access specific data. This is key in DTs where different users or systems may have varying levels of access to sensitive data.

4.1.1 Security Roles in ERDT Models

To improve the management of security constraints, we propose the implementation of security roles to enhance and simplify access control to the system. Instead of defining security constraints directly on each individual interface, security roles can be specified and used in the views. This approach provides a centralised mechanism for managing access rights and allows roles to be applied uniformly across different entities and views, smoothing the access control process. Roles can cover permissions to access specific entity operations and can be associated with multiple views.

Role-based security in ERDT models improves both security and maintainability, allowing access rights to be managed uniformly and efficiently across entities and views. This improvement makes ERDT especially applicable to dynamic environments where different user groups may need different levels of access to real-time data and operations.

4.2 Abstract Syntax

A *view* in an ERDT model is defined as a high-level query abstraction that interacts with multiple entities and their relationships by invoking operations from their respective entity interfaces. The view is a function that aggregates, filters, or transforms the data fetched from these entities, while applying the security constraints defined by the entity interfaces.

Let:

- $E = \{e_1, e_2, ..., e_n\}$ be the set of all entities in the ERDT model.
- $I(e) = \{o_1, o_2, ..., o_m\}$ be the set of interface operations defined for entity $e \in E$, where each operation o_i is a function $o : P \rightarrow A$, where P is the set of parameters, and A is the set of attributes or results for the operation.
- $S(o)$ is the set of security constraints for operation o_i, restricting which users $u \in U$ can invoke o_i.

A view $v \in V$ in the ERDT model is then a function that performs system-wide queries by linking the results of multiple entity operations while applying security and filtering conditions.

The formal definition of a view v is as follows:

$$v = f(\{e_k | e_k \in E, o_i \in I(e_k), C(o_i(p_j)), u \in S(o_i)\}) \quad (1)$$

where:

- $e_k \in E$ represents an entity involved in the view.
- $o_i \in I(e_k)$ is an interface operation of the entity e_k, and $p_j \in P$ are the parameters passed to that operation.
- $C(o_i(p_j))$ represents filtering conditions applied to the result of the operation $o_i(p_j)$, such as specific status or thresholds.
- $S(o_i)$ is the set of security restrictions and $u \in S(o_i)$ ensures that the user u has permission to access the operation o_i.
- f is an aggregation or transformation function that operates on the results.

4.3 Concrete Syntax

To facilitate the creation and management of views in the ERDT model, we propose a DSL that allows researchers and practitioners to define views in a clear and intuitive manner. This language provides a structured format, making it accessible for users to specify permissions, parameters, and query logic without needing in-depth knowledge of the underlying entity interfaces.

The definition language follows a straightforward format that emphasises clarity. The general structure can be described as shown in Listing 1.

```
view ViewName on EntitySet:
    permissions: <permission conditions>
    parameters: <parameter list>
    query: <filtering conditions>
```

Listing 1: View Definition.

This DSL is defined by a main *view* component that starts with the name given to the view. Then, it specifies with a *on* keyword which Entity Set the view acts on.

Following this definition, we find several attributes:

- *Permissions*: Declares the conditions that must be evaluated as true for the view to return results. We can use the method *canAccess* with an specific Entity Set interface to check if the current user has permissions to invoke it. In addition, the method *hasRole* can be called with a specific role to check if the current user belongs to it. These methods can be invoked and combined using a Boolean condition and operators like and/or.
- *Parameters*: Specifies the entry-point parameters that the user can pass to the view to dynamically limit the results through the following defined query. These parameters must be typed with the Value Set to which they belong and must be separated by a comma.
- *Query*: Specifies the query that will be applied over the Entity Set instances. An entity will be returned if this query is evaluated as true for that instance. Interfaces can be used to access entity attributes; only interfaces existing in the Entity Set definition specified in the view can be used. Value Sets instances and parameters specified in the view can also be used to be compared with the entity attributes through comparison operators. Normal operations include equality comparison and greater/lower than comparisons when the data types allow it. Custom comparison operators can be defined as custom methods that receive values and return a Boolean value at a higher level, and later can be implemented in the model transformation phase.

An example of a view that filters trucks by status can be seen in Listing 2

```
view TruckByStatus on Truck:
    permissions:
        canAccess(Truck::getStatus)
    parameters: Status status
    query:
        getStatus() = status
```

Listing 2: View Example.

This approach allows users to create views that encapsulate complex querying logic while ensuring the security policies established in the ERDT model. This abstraction improves usability and encourages the reuse of views in various contexts contributing to the overall efficiency and maintainability of the DT system. We can see more examples of view definitions in section 5.

5 PROOF OF CONCEPT VALIDATION

In this case study, previously commented in section 3, we show part of the logistics operations of Mercadona where a production-ready fleet management system controls a large fleet of delivery trucks. This system is built on a combination of data sources including a traditional relational database management system (RDBMS) and different proprietary third-party software solutions. These heterogeneous data sources make the architecture complex and depends on interconnected tables representing logistics entities such as trucks, drivers, routes, and sensor data from deliveries.

Mercadona's logistics team accesses operational data by executing SQL queries on the RDBMS, often through UI applications like Metabase (Metabase, 2024). However, they also interact with other systems that manage real-time data feeds, such as GPS tracking and maintenance logs. This multi-source environment increases the complexity of queries since a mix of database schemas and third-party data structures need to be integrated. Also, the user experience is affected since operational staff at Mercadona are not always SQL experts, creating a barrier to utilizing the system effectively. Thus, a more advanced approach to data querying and aggregation is needed.

Mercadona's fleet management system presents a significant opportunity for improvement. By transitioning to an ERDT-based model with query views, the system can offer more intuitive data access, better real-time insights, and greater alignment with operational requirements. Query views will enable non-technical users to easily access relevant, high-level information without the need to understand the low-level data structure or perform complex SQL queries.

This transition will not only simplify operations but also improve the logistics team's ability to monitor and manage the fleet efficiently ensuring faster decision-making and improved overall performance for Mercadona's delivery operations.

We define the following roles for accessing the DT entity views.

- Fleet Manager Role: A role with permissions to access views that provide detailed truck location and status information such as getLocation(), getStatus(), or getLocationHistory().
- Maintenance Staff Role: A role with permissions to access only vehicle health and diagnostic data, assuring that sensitive location data remains restricted.

To efficiently manage the truck fleet, we define

several views that aggregate data from multiple entities and apply security constraints. These views are aligned with the common queries currenlty done by fleet managers. We show 2 examples below.

TrucksInPolygon View. This view identifies trucks that are currently located within a specified polygonal geographic area by querying the 'Truck' entity's 'getLocation()' operation and filtering based on whether each truck's coordinates fall within the defined polygon.

```
view TrucksInPolygon on Truck:
    permissions:
        canAccess(Truck::getLocation)
    parameters: Polygon polygon
    query:
        polygon.contains(
            getLocation()
        )
```

where:

- The *query* filters trucks based on whether their location is within the specified polygon.
- The *permissions* ensure that the user has access rights to view each truck's location data.

This view allows users with appropriate permissions to monitor trucks within specific geographic areas, such as delivery zones or restricted regions.

TrucksIdleMoreThan15Min View. This view identifies trucks that have been idle (i.e., stationary at the same location) for more than 15 minutes by using the 'Truck' entity's 'getLocationHistory(from, to)' operation, which returns location and timestamp pairs over a specified time range. The view analyzes this history to detect if the truck has remained idle throughout the period.

```
view TrucksIdleMoreThan15Min on Truck:
    permissions:
        hasRole(fleetManagerRole)
    parameters: TimeStamp currentTime
    query:
        getLocationHistory(
            currentTime - 15m,
            currentTime
        ).unique().count() = 1
```

where:

- The *query* checks whether the truck has been in the same location throughout the specified time range, indicating idleness for over 15 minutes.
- The *permissions* ensure that only users with the 'fleetManagerRole' can access this view.

This view allows fleet managers to detect trucks that have been idle for extended periods, enabling intervention to optimize fleet operations.

6 DISCUSSION

As we have shown, the ERDT model provides a solid framework for modelling DTs, with improvements in development and operations.

- Greater abstraction and modularity: ERDT models encapsulate complex interactions and data through entities and interfaces. They allow developers to work with high-level concepts without the need to manage complex details, simplifying development and maintenance and offering a distinct advantage in environments involving complex data interactions.
- Simplified query creation: Queries in ERDT are modelled as views that represent real-world operations like tracking truck locations or detecting idle vehicles. These views are designed to be reusable and to facilitate data access.
- Enhanced security and access control: ERDT views allow for more granular control of data access, ensuring that security policies are enforced more effectively. This ensures a high level of security throughout the system even in environments with sensitive data sources.
- Ease of use and operational efficiency: Predefined ERDT views allow operational staff to easily access information without the need for technical expertise as they abstract away the technical complexities.

As an example, in our case study, current SQL-based solutions presented challenges in managing third-party data sources and applying security with the necessary granularity. With ERDT, these problems were solved through views, which showed a significant improvement in operational efficiency.

In summary, ERDT models offer a robust and secure solution suitable for dynamic and complex environments such as fleet management.

7 CONCLUSION AND FUTURE WORK

In this article we have presented an extension of the ERDT model by introducing views as a modelling tool for querying DTs. We have also shown how this solution addresses the challenges of complex DT implementations by showing a real case study at Mercadona. Contributions include a flexible and secure query mechanism and a DSL that simplifies the connection between complex data structures and operational needs.

Our ongoing work focuses on the development of a model transformation framework to automate the generation of code from ERDT models and views. In addition, a reference architecture will be proposed covering all layers from physical sensors to user interfaces with special attention to aspects such as security and scalability.

The ERDT model represents an important step forward in the development of DT. It provides a solid basis for building complex and secure systems. As DTs continue to evolve and become more widespread in different sectors, the need for efficient query mechanisms will grow. This work contributes to meeting that need and opens up new opportunities for research and practical application.

ACKNOWLEDGEMENTS

Author Pedro Valderas was financed by Project PID2023-146224OB-I00 founded by MICIU/AEI/ 10.13039/501100011033, FEDER, and the UE.

REFERENCES

Akehurst, D. H. and Bordbar, B. (2001). On querying uml data models with ocl. In Gogolla, M. and Kobryn, C., editors, *«UML» 2001 - The - The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, page 91–103, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bordeleau, F., Combemale, B., Eramo, R., van den Brand, M., and Wimmer, M. (2020). Towards model-driven digital twin engineering: 1st international conference on systems modelling and management, icsmm 2020. *Systems Modelling and Management - 1st International Conference, ICSMM 2020, Proceedings*, page 43–54.

Carrión, E., Pastor, Ó., and Valderas, P. (2025). Conceptual modelling method for digital twins. In Maass, W., Han, H., Yasar, H., and Multari, N., editors, *Conceptual Modeling*, page 417–435, Cham. Springer Nature Switzerland.

Chen, P. P.-S. (1976). The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36.

Digital Twin Consortium (2024). Digital twin definition.

Espinazo Pagán, J. and García Molina, J. (2014). Querying large models efficiently. *Information and Software Technology*, 56(6):586–622.

Gogolla, M. and Richters, M. (1998). On constraints and queries in uml. In Schader, M. and Korthaus, A., editors, *The Unified Modeling Language*, page 109–121, Heidelberg. Physica-Verlag HD.

Habela, P., Kaczmarski, K., Stencel, K., and Subieta, K. (2008). Ocl as the query language for uml model execution. In Bubak, M., van Albada, G. D., Dongarra, J., and Sloot, P. M. A., editors, *Computational Science – ICCS 2008*, page 311–320, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jia, W., Wang, W., and Zhang, Z. (2022). From simple digital twin to complex digital twin part i: A novel modeling method for multi-scale and multi-scenario digital twin. *Advanced Engineering Informatics*, 53:101706.

Jones, D., Snider, C., Nassehi, A., Yon, J., and Hicks, B. (2020). Characterising the digital twin: A systematic literature review. *CIRP journal of manufacturing science and technology*, 29:36–52.

Kirchhof, J. C., Michael, J., Rumpe, B., Varga, S., and Wortmann, A. (2020). Model-driven digital twin construction: synthesizing the integration of cyber-physical systems with their information systems. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '20, page 90–101, New York, NY, USA. Association for Computing Machinery.

Kleppe, A. (2008). *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Pearson Education.

Kolovos, D. S., Rose, L. M., Matragkas, N., Paige, R. F., Guerra, E., Cuadrado, J. S., De Lara, J., Ráth, I., Varró, D., Tisi, M., and Cabot, J. (2013). A research roadmap towards achieving scalability in model driven engineering. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*, BigMDE '13, page 1–10, New York, NY, USA. Association for Computing Machinery.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344.

Metabase (2024). Metabase: The easy way for everyone in your company to ask questions and learn from data.

Schroeder, G. N., Steinmetz, C., Rodrigues, R. N., Henriques, R. V. B., Rettberg, A., and Pereira, C. E. (2021). A methodology for digital twin modeling and deployment for industry 4.0. *Proceedings of the IEEE*, 109(4):556–567.

Stein, D., Hanenberg, S., and Unland, R. (2004). Query models. In Baar, T., Strohmeier, A., Moreira, A., and Mellor, S. J., editors, *«UML» 2004 — The Unified Modeling Language. Modeling Languages and Applications*, page 98–112, Berlin, Heidelberg. Springer Berlin Heidelberg.

Tao, F., Xiao, B., Qi, Q., Cheng, J., and Ji, P. (2022). Digital twin modeling. *Journal of Manufacturing Systems*, 64:372–389.

Tao, F., Zhang, H., Liu, A., and Nee, A. Y. C. (2019). Digital twin in industry: State-of-the-art. *IEEE Transactions on Industrial Informatics*, 15(4):2405–2415.