Integrating Flow and Structure in Diagrams for Data Science

Enea Vincenzo Napolitano
University of Naples Federico II
Italy

Elio Masciari
University of Naples Federico II
Italy

Carlos Ordonez
University of Houston
USA

***Abstract*—In Data Science, data modeling (including relational databases and big data) has traditionally used Entity-Relationship (ER) diagrams to represent structural characteristics of data. However, ER diagrams lack the capability to capture the flow and transformation of data through analytic pipelines, which are essential to manage modern data science workflows. On the other hand, flowcharts have been used for decades to describe the processing. Based on this motivation, this paper provides a historical perspective identifying the limitations of employing ER diagrams and Data Flow Diagrams (DFDs), separately, emphasizing the need to integrate both solutions. We examine established diagram notations and design models, including traditional models such as UML, ER, DFD, BPMN, and FLOWER. Our literature analysis suggests that integrative diagram approaches can provide a more intuitive and comprehensive understanding of data collection, data integration, and data transformation for big data analytics in the future.**

I. Introduction

Entity-Relationship (ER) diagrams have long been the dominant method for data modeling, offering a robust framework for the detailed and conceptual representation of data structures. ER diagrams serve to define entities, attributes, and the relationships between them in a manner that is effective and comprehensive, thereby establishing them as a cornerstone of traditional data modeling practices. However, they are primarily concerned with static structural representation and do not adequately capture the flow and transformation of data across processes. This limitation is made apparent when considered in conjunction with Data Flow Diagrams (DFDs), which emphasize the movement and processing of data, thereby underscoring the necessity for a more integrated approach in the field of modern data science and analytics.

In the context of contemporary data science processes, there is a discernible shift away from explicit data modeling towards a more pronounced emphasis on data manipulation and analysis. This transition can be attributed, at least in part, to the advent of modern tools such as IBM Workflow Diagram Software, Pentaho, ERWin, Lucidchart, and Tableau. These tools frequently employ their own proprietary visualizations and notation, rather than established standards such as ER or DFD. These tools prioritize flexibility and user-friendly interfaces, but may not fully capture the complexities of both data structure and flow in a unified manner.

Despite the prevalence of such tools, there is a continued need for effective data-modeling practices. In recent years, a number of methods have been proposed with the aim of enhancing the effectiveness of ER diagrams. These include the Unified Modeling Language (UML), DFDs, and the Business Process Model and Notation (BPMN). Each of these modeling techniques serves a distinct purpose and is beneficial at different stages of system development and analysis. However, these tools often operate in isolation, lacking the integration necessary to provide a comprehensive view of both data structure and flow.

In order to address this gap in the available tools, it may be beneficial to consider combining the structural representation of entity relationship diagrams with the dynamic visualization of data flow diagrams. The objective of this article is to emphasize the value of such integrated solutions in the field of data science, where an understanding of both the structure and flow of data is essential for deriving meaningful information.

II. Background

This section presents the theoretical background required to understand ER diagrams and DFDs.

A. Theoretical Concepts of the Entity-Relationship Model

The ER model is based on several foundational theoretical concepts, which provide a formal framework for data modeling. Once polished, an ER diagram is often mapped to a relational database schema, where the entities, attributes, and relationships are translated into tables, columns, and constraints that adhere to relational database principles.

1) Entities and Entity Sets: An entity set E is a collection of similar entities, which can be represented as:

$$E = \{e_1, e_2, e_3, \ldots, e_n\}$$

where e_i represents an individual entity in the set. Each entity is considered a unique instance of the real-world object or concept the entity set represents.

2) Attributes and Attribute Sets: Each entity in an entity set has attributes that describe its properties. These attributes can be formally defined as a function:

$$A : E \to V$$

where V is the set of possible values that the attribute can take.

Domain: For each attribute, its domain defines the set of permissible values. Formally, the domain D of an attribute A is a subset of V, specifying the valid range or set of values:

$$D \subseteq V$$

3) Relationships and Relationship Sets: A relationship set R is a collection of relationships, each involving entities from potentially different entity sets. It can be defined as follows.

$$R = \{(e_1, e_2, \ldots, e_k) \mid e_i \in E_i\}$$

where each e_i is an entity from a corresponding set of entities E_i. Relationships are crucial for illustrating how entities interact with one another in a model.

4) Keys:

- **Primary Key**: A primary key is a minimal set of attributes that uniquely identifies each entity within an entity set. Formally, if E is an entity set and $K \subseteq A$ is a set of attributes, then K is a primary key if and only if:

  $$\forall e_i, e_j \in E,\ (K(e_i) = K(e_j) \implies e_i = e_j)$$

  This ensures that no two distinct entities in the set share the same value for the primary key attributes, thereby providing uniqueness.
- **Foreign Key**: A foreign key (FK) is an attribute or a set of attributes in one entity set that serves as a reference to a primary key (PK) in another (or the same) entity set. Formally, if E_i and E_j are two entity sets and $FK \subseteq A_i$ (attributes of E_i) is a foreign key that references the primary key $PK \subseteq A_j$ (attributes of E_j), then for every $e_i \in E_i$, there exists an $e_j \in E_j$ such that:

  $$FK(e_i) = PK(e_j)$$

  In relational databases, this foreign key constraint is known as an *inclusion dependency*, which ensures that the values in the foreign key columns of one table must match the values in the primary key columns of another table. This concept is fundamental to maintaining relational integrity between tables.

5) Cardinality Constraints: Cardinality constraints specify the number of entities that can participate in a relationship. If R is a relationship set that involves entity sets E_i and E_j, then cardinality constraints can be expressed as:

$$\text{cardinality}(R, E_i) = (m, n)$$

where m and n represent the minimum and maximum number of times an entity from E_i can participate in the relationship R.

B. Data Flow Diagrams

Data Flow Diagrams (DFDs) are graphical representations used to model the flow of data within a system, highlighting the processes, data stores, and external entities involved. They are useful for both system analysis and design, providing a clear picture of how data moves through a system and how it is transformed by various processes.

Mathematically, DFDs can be understood within the framework of graph theory, where a DFD is represented as a directed graph $G = (V, E)$. In this graph, V represents the set of vertices and E represents the set of directed edges. Each vertex $v \in V$ can be classified into one of the following types:

- **Process vertices** ($P \subset V$): These vertices transform incoming data into outgoing data. They are often depicted by circles or rounded rectangles and represent operations or computations performed on data. Processes define the system's functionality.
- **Data store vertices** ($D \subset V$): These represent locations where data is stored and are usually drawn as open-ended rectangles or parallel lines. Data stores act as repositories where information is held for later use, and they interact with processes by either providing input data or receiving output data.
- **External entity vertices** ($X \subset V$): These denote the sources or destinations of external data to the system and are represented by rectangles. External entities are outside the system boundaries but interact with it by sending data to or receiving data from processes. They are crucial for understanding how the system interfaces with its environment.

The directed edges E, represented as arcs $(u, v) \in E$, signify the data flow between vertices. These edges are labeled with the data that are transmitted, providing clarity on what information is moving and how it is being transformed. For an edge $(u, v) \in E$, it indicates that data flow from vertex u to vertex v.

To represent the structure of a DFD more rigorously, the graph $G = (V, E)$ is often shaped like a flow network, consisting of a source vertex, intermediate processing vertices, and a sink (destination) vertex. The source vertices ($S \subset V$) represent the starting points of data flows (typically external entities), while the sink vertices ($T \subset V$) represent the endpoints (often data stores or external entities receiving data). The intermediate vertices ($I = V \setminus (S \cup T)$) represent the processes that transform or route data as it flows through the system.

For a more formal computational analysis, the structure and flow of a DFD can be described using incidence matrices or adjacency matrices, which are common tools in graph theory. For a given DFD graph $G = (V, E)$, its adjacency matrix A is defined as:

$$A_{ij} = \begin{cases} 1 & \text{if there is a directed edge from vertex } v_i \text{ to } v_j, \\ 0 & \text{otherwise.} \end{cases}$$

This matrix formulation facilitates computational analysis and simulation of data flows. For example, by analyzing the powers of the adjacency matrix, one can determine the transitive closure of data flows, identifying all possible indirect paths between vertices.

Moreover, DFDs can be hierarchical, meaning that complex processes can be broken down into more detailed subdiagrams, known as "leveled" DFDs. This allows for a layered approach to system modeling, starting from a high-level overview and drilling down into finer details as needed. Each level must maintain consistency with its parent level, ensuring that all data inputs and outputs remain compatible across different abstraction layers.

C. BPMN

Business Process Model and Notation (BPMN) is a standardized graphical notation used to model business processes. BPMN is designed to bridge the gap between business process design and implementation by providing a clear and intuitive way to represent workflows.

Core Elements:

- **Flow Objects:**
  - *Events:* Represent the start, intermediate, and end points of a process.
  - *Activities:* Tasks or work performed within the process, depicted as rectangles with rounded corners.
  - *Gateways:* Decision points that control the divergence or convergence of process flows.
- **Connecting Objects:** Arrows that define the sequence flow, message flow, and associations between flow objects.
- **Swimlanes:** Used to group activities by actor or role, represented as pools and lanes.
- **Artifacts:** Additional information, such as data objects or annotations, to provide context.

III. Historical Perspective

This section examines the history of key data modeling methods and how they have evolved over time, highlighting their strengths, limitations, and contemporary relevance.

A. Data Structure

1) ER Diagram: In 1976, Peter Chen introduced ER diagrams as a method for conceptual modeling of data. This approach was developed to address the limitations of previous modeling methods by providing a more natural and visual way of representing relationships between entities in computer systems. The ER model rapidly became the de facto standard for database design due to its capacity to represent complex relational scenarios in a transparent and comprehensible manner [1].

One of the key strengths of ER diagrams is their flexibility, which allows for the representation of a virtually unlimited range of data structures. It can be safely assumed that any object in the world can be identified in some way, whether by position, index in an array, or other attributes. This makes ER diagrams highly adaptable. Furthermore, the process of ER modelling serves to abstract away the complexities of programming and computer architecture, focusing purely on the data structure and its relationships without the need for concerns about implementation details.

Over time, ER diagrams have been widely adopted in different fields, including relational database design, information system design, and the integration of heterogeneous systems. As more complex data environments have emerged, such as object-oriented databases and NoSQL databases, the ER model has been adapted to represent non-relational data structures. However, its primary application remains in the context of traditional relational databases [2]. The advent of data lakes and AI data repositories has given rise to new challenges in the efficient storage of diverse data formats (e.g. text files, web pages, blogs) that do not map neatly to SQL-based structures. These developments illustrate the evolving role of entity-relationship diagrams in modern data management solutions.

2) UML: The Unified Modeling Language (UML) was developed in the early 1990s in response to the increasing complexity of software systems. The main creators of UML were Grady Booch, Ivar Jacobson, and James Rumbaugh. UML was born out of the need to unify various existing modeling methods such as OMT (Object Modeling Technique), the Booch method, and Objectory. The first official version of UML was released in 1997 under the auspices of the Object Management Group (OMG). Since then, UML has become an international standard for modeling software systems, providing a comprehensive range of diagrams for software design, analysis, and documentation [3].

Unlike ER diagrams, UML introduces the concept of "methods" attached to "classes," which can be thought of as "entities with methods." This distinction is significant because, while ER diagrams focus purely on data structure and relationships, UML provides a way to model the behavior of the system in addition to its structure. In software development, where the source code tends to be structured around functions or methods, UML's object-oriented approach offers a more holistic view of the system's architecture.

As software development methods have evolved, UML has seen a change in its applications. In the early 2000s, it was mainly used in the context of formal documentation-based software development methods, such as the waterfall model. However, with the rise of agile methodologies, the use of UML has been somewhat reduced, favoring simpler diagrams and greater iteration. Despite this, UML remains a key tool in areas where detailed documentation is required, such as industrial software engineering and embedded systems [4].

Figure 1 illustrates an example of a UML representation of the classes that correspond to an ER model.

B. Flow

1) Data Flow Diagram: The Data Flow Diagram (DFD) was first introduced by Tom DeMarco in his publication, 'Structured Analysis and System Specification', released in 1979. DFDs have become a principal tool for modeling data

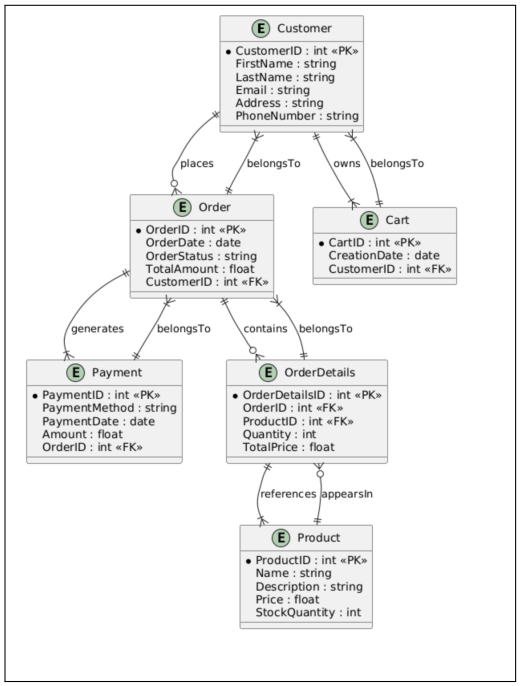

Fig. 1. ER diagram in UML notation of an order management system for an online shop.

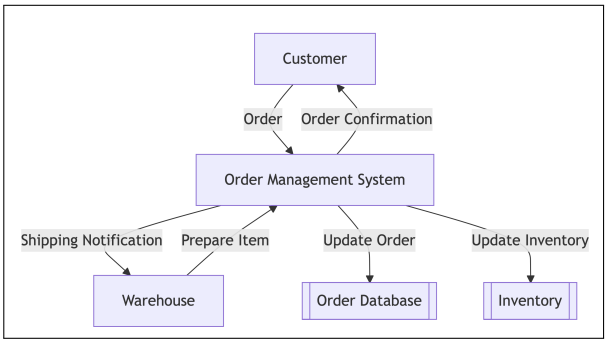

Fig. 2. An example of a level 0 DFD for an online shop.

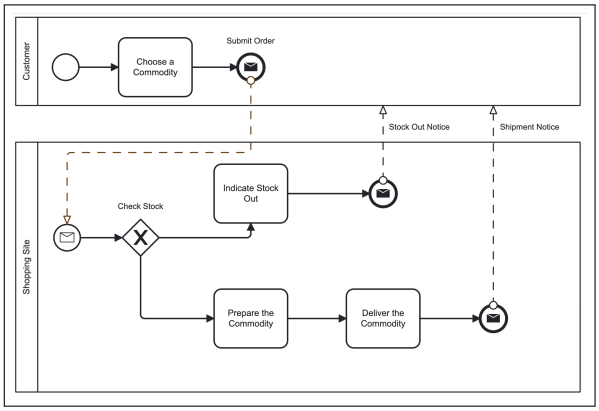

Fig. 3. BPMN example of an online shop.

flows in computer systems. This type of diagram represents the flow of data between various parts of a system, with the objective of facilitating the understanding and analysis of system requirements. A DFD is formally defined as a directed graph $G = (V, E)$, where V represents the set of vertices (processes, data stores, external entities) and E represents the set of directed edges (data flows). Over time, the formal definition of DFDs has been refined, but their core conceptual framework remains the same.

DFDs were widely used during the 1980s and 1990s, especially in information system development projects, and are still used today in specific contexts [5]. They continue to be valuable tools in the analysis of legacy systems, the documentation of existing systems, and the education of software engineers due to their simplicity and visual clarity [6]. Despite the decline in their use for new systems development, DFDs remain relevant for specific applications.

Figure III-B1 illustrates an example of a DFD.

2) BPMN Diagram: The BPMN was developed in response to the need for a standardized representation of business processes. BPMN was initially developed by the Business Process Management Initiative (BPMI) in 2004 and was subsequently incorporated into the OMG, which released version 2.0 in 2011. BPMN was designed with the intention of being comprehensible to both business analysts and technical developers, with the objective of bridging the gap between the design of business processes and their implementation in information systems[7], [8].

It is worth noting that BPMN appears to have been inspired more by the conceptual framework of Data Flow Diagrams (DFDs) than by Entity-Relationship (ER) Diagrams. This is evident in its emphasis on modeling the flow of processes, tasks, and decisions within a system, which aligns more closely with the principles of DFDs that prioritize the movement and transformation of data rather than the static relationships between entities, as in ER diagrams.

BPMN is often used in conjunction with ER diagrams to provide a more complete representation of both business processes and data structures. The release of BPMN version 2.0 saw significant enhancements, including support for collaborative processes and the ability to model complex events. This has positioned BPMN as a versatile tool for process modeling in dynamic and global business contexts [9].

Figure III-B2 provides an example of BPMN.

C. Current Solutions

Recent years have seen the development of various methods to address the limitations of traditional data modeling techniques and better integrate data flow and structure modeling.

One notable approach is DBPMN (Data-aware and Decision-aware BPMN), which builds on BPMN and DMN S-FEEL (Simple-Friendly-Enough-Expression-Language) to create a more formal and native definition of business process models. This approach allows for the encoding of both data- and decision-aware processes, providing a more comprehensive and formalized representation of business workflows that includes both process flow and decision logic. This integrated

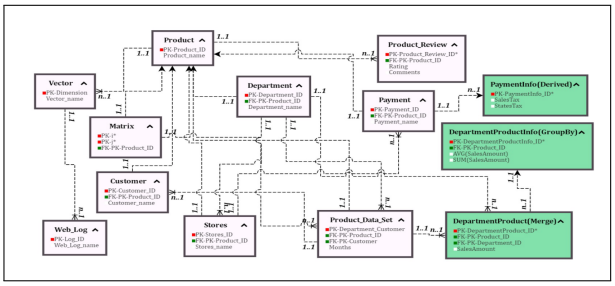

Fig. 4. An example of a FLOW + ER.

model is particularly valuable for capturing the complex dynamics of modern business processes, especially in data-driven environments [10].

Another innovative method is the hybrid process modeling approach, which combines BPMN, CMMN (Case Management Model and Notation), and DMN to create a more holistic and efficient representation of business processes. By integrating these three notations, organizations can achieve a more structured and realistic representation of their processes, simplifying the modeling process and improving the clarity of process documentation [11].

The landscape of BPMN extensions is also rich in various proposals that aim to improve the expressiveness and applicability of BPMN in different contexts. A systematic review of these extensions [12] highlights the various ways in which BPMN has been adapted to meet the needs of specific industries and use cases, from incorporating temporal constraints to enabling data-driven process modeling.

In the area of data representation, the emergence of data lakes and AI data repositories has reshaped how we think about data storage and management. Modern data warehouses have evolved into data lakes [13], capable of storing semi-structured and unstructured data formats such as JSON, text files, web pages, blogs and tweets, which cannot be efficiently managed using traditional SQL databases. The FLOWER method (FLOW + ER)[14] aims to synthesize the strengths of these disparate approaches, providing a comprehensive model that unifies data flow and structural representation, capturing diverse data types and their movement within systems.

Figure III-C illustrates an example of a FLOWER. It is evident that the addition of arrows (representing the data flow) to UML class diagrams (which are structurally analogous to ER diagrams with augmented methods) is a relatively straightforward process. In contrast, the incorporation of structured elements (such as entities with attributes) into process-centric diagrams, including BPMN and DFD, is a more complex undertaking.

In practice, however, many data processing tasks, especially in environments such as Python, still rely on simple, unstructured data formats such as CSV files without an underlying data model. While this approach offers simplicity and flexibility, it often lacks the rigor and consistency provided by formal data models, thereby underscoring the continued need for advanced modeling solutions in the evolving landscape of data science and business process management.

IV. Current Landscape and Future Directions of Flow and Data Structure

The evolving landscape of data science and AI highlights the critical need for data modelling approaches that seamlessly integrate data flow and structure. Traditional techniques such as ER diagrams, UML, DFDs and BPMN provide valuable frameworks for capturing business processes and relationships. However, they often fall short in supporting the complex, unstructured, and multidimensional data formats prevalent in modern AI applications and data science workflows, such as vectors, matrices, JSON, and other non-relational data types. We argue for a renewed emphasis on comprehensive data modelling frameworks that go beyond the limitations of traditional methods to support both business and mathematical processing. These frameworks should be able to represent the complex data structures and flows that are fundamental to effective data-driven decision making and advanced analytics.

The comparison is presented in Table I, which highlights the formal definition, main elements, and clarity of each method, among other factors.

Table II provides a summary of the advantages and limitations of each method, providing further information on the relative strengths and weaknesses of these approaches.

In conclusion, a significant challenge today is the widespread use of code-centric approaches, particularly in environments such as Python, where data manipulations are performed without a coherent underlying data model. This can lead to inefficiencies, lack of scalability, and difficulties in maintaining and interpreting data workflows. We believe that data models are essential not only for traditional data management, but also for managing the complexity of modern big data environments, such as data lakes and AI data repositories. Future modelling solutions should strive to integrate the strengths of existing techniques while accommodating different data types and processes, enabling more robust systems that can evolve with advances in data science and AI.

References

[1] P. P.-S. Chen, "The entity-relationship model—toward a unified view of data," *ACM transactions on database systems (TODS)*, vol. 1, no. 1, pp. 9–36, 1976.

[2] R. Elmasri, "Fundamentals of database systems seventh edition," 2021.

[3] G. Booch, I. Jacobson, J. Rumbaugh *et al.*, "The unified modeling language," *Unix Review*, vol. 14, no. 13, p. 5, 1996.

[4] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2018.

[5] T. DeMarco, "Structured analysis and system specification," in *Software pioneers: contributions to software engineering*. Springer, 2011, pp. 529–560.

[6] Q. Li and Y.-L. Chen, "Data flow diagram," in *Modeling and Analysis of Enterprise and Information Systems*. Springer, 2009, pp. 85–97.

[7] S. A. White, "Introduction to bpmn," *Ibm Cooperation*, vol. 2, no. 0, p. 0, 2004.

[8] M. Von Rosing, S. White, F. Cummins, and H. De Man, "Business process model and notation-bpmn." 2015.

[9] J. Recker, "Opportunities and constraints: the current struggle with bpmn," *Business Process Management Journal*, vol. 16, no. 1, pp. 181–201, 2010.

TABLE I
COMPARISON OF DATA MODELS WITH FOCUS ON DATA SCIENCE ACTIVITIES

	UML Activity Diagram	BPMN	DFD	ER Model + UML	ER Model + Flow
Fundamental Aspects Definition	A directed graph $G = (A, T)$, where A is a set of activities, and T is a set of transitions (directed edges) between them, with optional decision nodes [15]	A tuple (E, A, G), where E is a set of events, A is a set of activities, and G is a set of gateways, with directed edges representing the control flow between these elements [16]	A directed graph $G = (P, D, F)$, where P is a set of processes, D is a set of data stores, and F is a set of data flows (edges) between them	A tuple (E, R), where E is a set of entities, each defined as a set of attributes, and R is a set of relationships, extended with UML class diagrams as sub-graphs $S(E, R, C)$, where C is a set of class associations	A tuple (E, R, F), where E is a set of entities, R is a set of relations, and F is a set of directed data flows, with F represented as directional arcs connecting E and R
Main Elements	Activities, transitions, decisions, swimlanes	Events, activities, gateways	Processes, data stores, data flows	Entities, relationships, attributes, classes	Entities, relationships, data flow arrows
Purpose	Model workflow and step-by-step activities in a process	Model business processes and data flows	Visualize data flow and transformations	Model data structure with integrated UML notation	Unify data flow and ER diagrams
Detail Level	Detailed workflow and activity sequences	Broad business process context	Focus on data flow	Detailed data structure with UML enhancements	Integrates data flow with structure
Data Transformation Focus	Sequence of activities and decisions	Integrated into business processes	Data flow representation	Before/after structure with class relationships	Data flow across ER entities
Data Science Operations	Models sequences of operations and decisions	Integrates with business operations	Clear transformation steps	Focus on structural changes, enhanced with UML	Merge, group by, derived transformations
Ease of Understanding	Requires familiarity with UML and workflow modeling	Easy for business users	Intuitive for data flows	Requires ER and UML knowledge	Requires ER knowledge, integrated flow
System Integration	Can be integrated with other UML diagrams	Broad process view	Data-focused, less on system integration	Focuses on data structure with UML linkage	Integrates flow and structure
Clarity	Defined by minimal $\|A\| + \|T\|$, maximizing readability through simplicity and linear flow where possible	Defined by clarity of control flow, optimizing for minimal complexity in $\|E\| + \|A\| + \|G\|$	Defined by minimal crossings in G and linear representation of data flows F	Defined by minimal entity overlap and clear separation of class and relationship diagrams	Defined by the minimal overlap of E, R, and F, optimizing for clear visual flow and structural integrity

TABLE II
COMPARISON OF DATA MODELING TECHNIQUES

	UML Activity Diagrams	BPMN	DFD	ER Model + UML	ER Model + Flow
Advantages	High detail and specificity for object-oriented systems; Clear visualization of complex processes	Widely understood in business contexts; Easy to communicate across different stakeholders; Effective for aligning technical processes with business goals	Clear representation of data flow; Simple and intuitive; Facilitates quick understanding of data flows	Enhanced structural modeling with integrated class relationships; Strong focus on data integrity and relationships	Comprehensive view of complex systems by integrating data flow and structure; Designed for data science, capturing common operations like merging and grouping; High compatibility with existing data management tools; Can be automatically generated from source code
Limitations	Requires significant familiarity with OOP and UML; Can be complex to implement and understand	Less detailed for technical processes; Extending BPMN for data science operations can introduce complexity and compatibility issues	Limited in capturing data structure; Focuses only on the movement of data, not on the structure	More complex due to UML integration; Requires dual expertise in ER modeling and UML; Less adaptable to dynamic data flow scenarios	Requires understanding of both ER and flow concepts

[10] M. De Leoni, P. Felli, and M. Montali, "Integrating bpmn and dmn: modeling and analysis," *Journal on Data Semantics*, vol. 10, no. 1, pp. 165–188, 2021.

[11] N. Passos and J. L. Pereira, "Business process modeling: how cmmn and dmn complement bpmn," 2018.

[12] K. Zarour, D. Benmerzoug, N. Guermouche, and K. Drira, "A systematic literature review on bpmn extensions," *Business Process Management Journal*, vol. 26, no. 6, pp. 1473–1503, 2020.

[13] A. A. Harby and F. Zulkernine, "From data warehouse to lakehouse: A comparative review," in *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 2022, pp. 389–395.

[14] C. Ordonez, S. Maabout, D. S. Matusevich, and W. Cabrera, "Extending er models to capture database transformations to build data sets for data mining," *Data & Knowledge Engineering*, vol. 89, pp. 38–54, 2014.

[15] M. Bures, B. S. Ahmed, and K. Z. Zamli, "Prioritized process test: An alternative to current process testing strategies," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 07, pp. 997–1028, 2019.

[16] P. Y. Wong and J. Gibbons, "Formalisations and applications of bpmn," *Science of Computer Programming*, vol. 76, no. 8, pp. 633–650, 2011.