Model-to-Model Transformation: From UML Class Diagrams to Labeled Property Graphs

Ana Leon[1,2*], Maribel Yasmina Santos[2], Alberto García[1], Juan Carlos Casamayor[1], and Oscar Pastor[1]

[1]Universitat Politècnica de València,
Camí de Vera S/N, 46022, Valencia, Spain
[2]ALGORITMI Research Centre, University of Minho
Campus de Azurém, 4800-058, Guimarães, Portugal

*To whom correspondence should be addressed; E-mail: aleon@vrain.upv.es.

Conceptual schemas are the basis to build well-grounded Information Systems, by representing the main concepts of a domain of knowledge, as well as the relationships among them. Since conceptual schemas focus on the concepts, they are independent of the specific technological platform used to implement them. This allows a single conceptual schema to be transformed into different platform-specific models according to the implementation requirements. This is a non-trivial process that is crucial for the performance and maintainability of the system, as well as for the accomplishment of the domain data requirements. Much research has been done on transforming conceptual schemas into relational data models. Nevertheless, less work has been done on transforming conceptual schemas into property graphs, a data structure indispensable to building appropriate and efficient systems based on graph databases. This work proposes a systematic approach to transform conceptual schemas, represented as UML class diagrams, into property graphs by using a set of transformation rules and patterns applied in a systematic way. Besides a practical example used to help the presentation of the proposed approach, the evaluation has been done by measuring different quality dimensions such as semantic equivalence, readability, maintainability, complexity, size, and performance.

Keywords - *Conceptual data models; Class diagrams; Property graphs; Graph databases*

1 Introduction

The use of Model Driven Development (MDD) strategies to perform transformations between Platform Independent Models (PIM) and Platform Specific Models (PSM) is the basis for building well-grounded Information Systems, and ensuring the maintenance of the conceptual integrity of the data that are going to be managed (Pastor, España, Panach, and Aquino (2008)). By conceptual integrity we mean that no inconsistencies are introduced during the proposed transformation because what is modeled in the PIM is accordingly represented in the PSM, synchronizing the different levels of abstraction that participate in the transformation process. In this work, we tried to fill the gap found in one of the most important tasks in any MDD process when moving from PIMs to PSMs: the model-to-model transformation involving property graph schemas.

Conceptual schemas are well-known PIMs that represent the main concepts of a domain of knowledge, as well as the relationships among them (Pastor and Molina (2007)). Since conceptual schemas focus on the concepts, they are independent of the specific technological platform used to implement them. Therefore, a single conceptual schema can be transformed into different PSMs according to the technological requirements. This is a non-trivial process that is crucial for the performance and maintainability of the system, as well as for the accomplishment of the domain data requirements. The more aligned with the requirements the model is, the more efficient the system will be. This efficiency can be measured not only in terms of query performance, but also in terms of the system's maintainability (i.e., readability, complexity, and size).

Much research has been done on transforming conceptual schemas into relational data schemas. Examples of these research works are Rahayu, Chang, Dillon, and Taniar (2000), and El Alami and Bahaj (2018). Nevertheless, less work has been done on transforming conceptual schemas to property graph schemas. This is because graph databases have been designed to be schema-less (unlike relational databases) and the modeling process has been incorrectly relegated to the background. If we want to store and query the data in a way that efficiently addresses the user's analytical requirements and be able to update the database as the requirements change, the use of appropriate modeling techniques and systematic model transformation methods must be the key tasks of the design process.

This work intends to be the first step in this direction, by proposing a systematic approach to transform conceptual schemas (represented as UML class diagrams) into property graphs by using a set of transformation rules and patterns applied in a systematic way. The transformation rules guide the process from the identification of the analytical requirements to the application of the patterns. The patterns express the several modeling alternatives that can be used when transforming classes and associations into nodes and edges of a property graph. We take advantage of UML Class Diagrams as a well-known and widely used language to deeply explain the approach proposed. It can help readers to understand how the concepts that are being managed can be applied by using design languages and data structures that are well-known in the Information Systems community in terms of Model-Driven Development. This work also serves as a first step for the generalization to other languages and data structures that we are considering as future work. For example, other UML diagrams such as activity, interaction, or use case diagrams related to functional aspects.

The research methodology used in this work is the Design Science Research Methodology for Information Systems (Hevner, March, Park, and Ram (2004); Peffers, Tuunanen, Rothenberger, and Chatterjee (2007)). This methodology is divided into six steps: Problem Identification and Motivation, Definition of Objectives, Design and Development, Demonstration, Evaluation, and Communication. According to this methodology, the main purpose of this work is to propose a systematic approach to transform UML Class Diagrams into Labeled Property Graphs (LPGs), filling the existing gap when performing PIM to PSM transformations for graph databases. Therefore, the main objectives to achieve are i) the identification of the core constructs used in UML to model a domain of interest, ii) the proposal of the modeling alternatives (patterns) to transform each construct into a property graph (considering the analytical requirements), and iii) the formalization of a systematic process based on a set of transformation patterns.

After the systematization of the process and the definition of the transformation patterns, we present a demonstration case to help the reader to understand the process in a familiar domain (a flight management system). Finally, the proposed process and transformation patterns are evaluated by comparing two competing LPGs and measuring different quality dimensions such as semantic equivalence, readability, maintainability, complexity, size, and performance. By measuring the semantic equivalence we ensure that both models preserve their intended behavior. By measuring the performance, we prove that models built considering the domain data requirements lead to more efficient queries. Finally, by measuring the readability, maintainability, complexity, and size, we show how aligned are the domain data requirements with the global maintainability of the system.

To achieve this goal, the paper is structured as follows. After this introduction, Section 2 describes the main characteristics of the UML Class Diagrams and the LPGs. Section 3 presents the related work and its limitations. In Section 4 we describe our proposal to transform UML Class Diagrams into LPGs, and in Section 5 we present a practical example of application. In Section 6, an evaluation is performed and Section 7 presents the conclusion and future work.

2 UML Class Diagrams and Labeled Property Graphs

Unified Modeling Language (UML) is a standardized modeling language developed to help system and software developers to specify, visualize, and document models, including the artifacts of software systems and business modeling (Jacobson & Booch, 2021). UML is platform-independent, which means that the specified models are independent of the specific technological platform used to implement the software or business system. Therefore, it is possible to work at a higher level of abstraction, ensuring that business functionalities are complete and correct, end-user needs are met, and system design supports the established requirements before coding. There are different types of diagrams specified in the UML standard (e.g., activity diagram, interaction diagram, and class diagram). In this work, we use the class diagram, widely used in the modeling of object-oriented systems. The basic elements of a class diagram are classes, attributes, and associations (Sparks (2001)). A class describes a set of objects that share the same features, constraints, and semantics (meaning). An attribute defines values that can be attached to the instances of a class. Finally, an association is a relationship between classes that is used to show that instances of classes could be either linked to each other or combined logically or physically into some aggregation.

A labeled property graph (LPG) is the result of translating a conceptual view of a domain into an integrated logical and physical data model. This model includes the knowledge of the main domain concepts, how they are related, and the way they should be physically organized in a graph database. Property graph's basic elements are nodes, labels, properties, and relationships (Robinson, Webber, & Eifrem, 2015). A node is often used to represent an entity of the application domain. Nodes can have one or more labels, as a way of introducing a certain level of classification and schematization. Nodes contain properties (key-value pairs). A relationship connects two nodes establishing the semantic context between them. They have a direction and a type and can include properties to highlight the characteristics of the relationships. The direction adds meaning to the representation, although the relationship is navigable in both directions. Figure 1 shows the differences between the basic elements of a UML class diagram and a property graph.

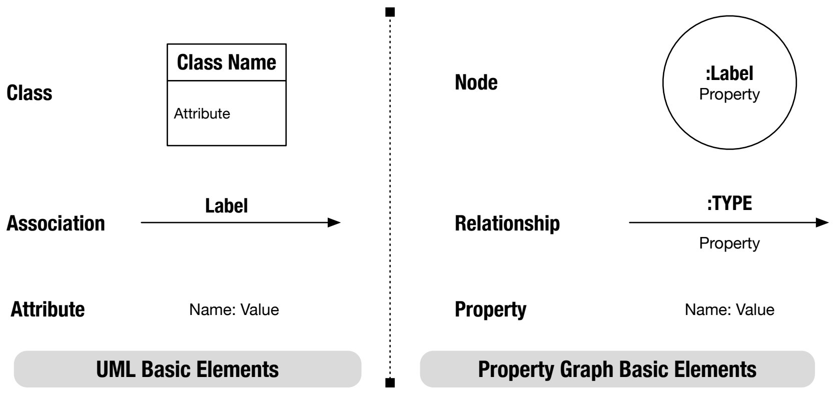

Figure 1: Basic elements of a UML Class Diagram and a Labeled Property Graph.

The transformation of UML Class Diagrams into LPGs is a complex process that must consider the analytical requirements and must be systematized to help the data engineer make the right decision for each scenario.

3 Related Work

The research on transforming PIMs into graph databases is focused on defining meta-models as an intermediate representation to facilitate the direct mapping between conceptual model elements and graph database elements. In Daniel, Sunyé, and Cabot (2016), the authors propose a framework that translates conceptual schemas expressed using UML Class diagrams to generate a meta-model that facilitates the integration of several kinds of graph databases. The proposed meta-model defines a possible common structure for graph stores, including elements such as vertex, edge, and property. With this meta-model, the proposed transformation maps classes to vertices, associations to edges, and so on. The work of De Virgilio, Maccioni, and Torlone (2014) uses the conceptual representation of the domain expressed in an Entity-Relationship (ER) model and proposes a strategy for the identification of a graph data model that minimizes the number of nodes, as it follows a compact design strategy. To avoid data inconsistencies that can occur when following compact strategies, the approach analyses all many-to-many relationships in the ER model. The approach mainly relies on the analysis of the relationships between the entities and their aggregation.

The work of Groger, Schwarz, and Mitschang (2014) proposes the Deep Data Warehouse for the flexible integration and enrichment of warehouse data and unstructured content. The proposed approach uses information-rich instance-level links that associate warehouse elements and content items in a graph-oriented structure. The authors use a conceptual linking model to obtain a logical schema that facilitates the direct mapping of links using properties on the edges of a property graph. In Akid, Frey, Ayed, and Lachiche (2022), the authors propose a set of rules to convert a multidimensional data model into a graph data model, suggesting a set of rules that enable the identification of two graph models having a star-like or a snowflake-like schema. The approach maps multidimensional data concepts (such as facts and dimensions) into graph concepts, transforming fact tables into fact nodes, dimension tables into dimension nodes, and relationships between fact and dimension tables into edges between the fact and dimension nodes. The works of Castelltort and Laurent (2014), Sellami, Nabli, and Gargouri (2020), and Gómez, Kuijpers, and Vaisman (2020) also address the proposal of rules or similar approaches to transform multidimensional data models into graph-oriented data models. Abdelhedi, Brahim, Atigui, and Zurfluh (2017) propose an approach based on transformation rules to translate UML conceptual models into NoSQL physical models, such as column, document, or graph implementations. In this approach, a generic target meta-model is proposed, which includes common features of these three NoSQL physical models, allowing the transformation between a source and a target meta-model. Nevertheless, the experimental work evaluated the physical model implemented in column and document data stores, but not in graph databases.

Regarding automated tools, diverse tools have been proposed for translating models into different structures, including graph-based structures (Ziemann, Hölscher, and Gogolla (2005), Huang, Duan, Sun, Lin, and Zhu (2016), Karagiannis and Buchmann (2018), Burzynski and Karagiannis (2020)). More recently, the work of Smajevic, Hacks, and Bork (2021) transforms ArchiMate models into knowledge graphs, providing a set of queries on the knowledge graph representation to detect Enterprise Architecture Smells, and the work of Glaser, Ali, Sallinger, and Bork (2022) introduces a method for model-based Enterprise Architecture Knowledge Graph construction, highlighting how ArchiMate models can be further enriched by Enterprise Architecture-specific and graph characteristics-based knowledge.

All these works agree on considering conceptual models useful to get a shared understanding of a domain. Nevertheless, they do a direct mapping between classes or entities to nodes (or vertices). Since the approaches tend to do a direct mapping between concepts, we consider that this is a gap in the literature, as different data models can represent the same application domain, but with different degrees of efficiency and efficacy when considering the analytical requirements to be met. Our proposal fills this gap, allowing the transformation to an LPG that is designed to ensure the performance and maintainability of the system, as well as the accomplishment of the domain data requirements. This last characteristic is a key contribution of the approach proposed in this paper, as it considers not only the semantics of the conceptual schema but also the role of each concept or entity in this schema, enhancing the analytical value of the property graph when supporting decision-making activities. Also, our work is based on a pattern-driven approach that formalizes the modeling concepts and the different possible transformations. A pattern is here understood as a formalisation of a problem that arises in a specific context, the proposal of a solution and when the solution can be applied (Blaha (2010)). As advantages of this approach, conceptual data modeling based on patterns allows i) reusability, as patterns can be reused; ii) knowledge transfer, as patterns capture design and development knowledge; iii) standardisation, as patterns provide a standardised solution; iv) quality, as patterns improve uniformity and documentation; and v) ease abstraction (Albdaiwi, Noack, and Thalheim (2014)).

4 From UML Class Diagrams to Property Graphs

As described in the related work section, there is a tendency to make a direct association between classes and nodes, attributes and properties, associations and relationships. Nevertheless, we believe this is not the best approach since the aim and the level of abstraction of both diagrams are different. Class Diagrams do not consider the performance of the system but focus on providing a sound structure to the data. Property Graphs structure the data based on the representation provided by the class diagrams but must ensure that the system is as efficient as possible in terms of query performance, data storage, and scalability. This requires following a series of steps, applied in a systematic way, to transform the Class Diagram into the most appropriate Property Graph.

4.1 The Transformation Process

A model-to-model transformation process is commonly based on identifying the elements of the original model that must be transformed, and on applying a pattern to represent them according to the destination model specifications. The fulfillment of the analytical requirements is a key aspect of our proposal and determines how the original elements are going to be transformed. To such an aim, the analytical requirements must be clear from the beginning, and the identification of the analytical characteristics of the original elements must be done before the application of the transformation patterns. According to this, the transformation process proposed in this work follows a series of systematic steps: i) Identification of the analytical requirements, ii) Identification of the analytical classes, iii) Identification of the most adequate patterns, iv) Application of the patterns, and v) Model refinement.

4.1.1 Identification of the analytical requirements

The identification of the analytical requirements is guided by the queries to be supported. This means that the transformation process is guided by the objective of obtaining a graph data model that considers the decision-making needs for an application domain that is modeled using UML Class Diagrams principles and concepts. Therefore, the main objective of this step is to get a shared understanding of the main concepts described in the Class Diagram and to clarify the queries the system must answer. For example, in a flight management system (FMS), the main concepts represented in a Class Diagram could be Flight, Booking, Passenger, and Airline. One of the interesting queries to be solved could be *which are the flights operated by Airline X?* The different queries will determine the analytical importance of each element of the Class Diagram and, therefore, the structure of the resulting property graph.

4.1.2 Identification of the analytical classes

Once the analytical requirements are clear, and the main concepts involved are determined, the classes and attributes of the Class Diagram that correspond to these concepts must be identified and classified according to the following analytical characteristics, proposed in the work of Galvão, Leon, Costa, Santos, and Pastor (2020):

1. Classes with high analytical value (C1): classes involved in the defined queries pointing to the main business processes with the key activities of the domain. In the FMS example, an analytical class could be Flight considering the query specified in the previous example.

2. Classes with high cardinality (C2): classes with increasing data volume, challenging the querying capabilities of a specific data structure. This information considers the number of instances expected for each class.

3. Classes with frequent access patterns (C3): classes frequently used, along with C1 classes, for answering the most common application domain queries. These classes are commonly involved in joining multiple domain entities. In the previous example, Airline could be classified as C3.

As can be seen, the analytical importance of each class is highly dependent on the defined queries, the application domain, and the background knowledge of the data engineering about the data and its evolution, in terms of volume. For the proposed approach, C1 and C3 classes are semantically different as C1 classes are usually queried to analyze the main business indicators or activities of the domain (*what happened or is happening?*), such as *how many flights ...? or which flights ...?*, while C3 classes are used to add semantics to these queries (*when or where happened?*). For example, *how many flights went to Airport JFK on Jan 21, 2021?*. This distinction is relevant in the model refinement step of the approach. C1 classes are meant to be the key concepts highlighted in the property graph. They should maintain their existence and relevance as the central nodes of the property graph, while C3 classes leave room for later optimization of the property graph by the data engineer considering, for instance, data volume issues. As an example, two different nodes could be merged into one node if this increases the overall performance of the system or better fits other systems' requirements. These concerns with the main concepts of the domain, as well as the associated data volume, guide the transformation process for the identification of an effective and efficient property graph.

4.1.3 Identification of the most adequate patterns

Once the analytically relevant classes have been identified, a set of systematic tasks can be applied, supported by the transformation patterns described afterwards in this work:

1. Represent C1 classes as nodes and the relationships with other C1 classes.
2. Represent C2 classes as nodes and the relationships with C1 and other C2 classes.
3. Represent C3 classes as nodes and the relationships with C1, C2, and other C3 classes.
4. Represent all the remaining classes and their associations with the other classes.

These steps consider that C1 classes should guide the identification of the property graph and its main concepts, C2 classes follow these introducing performance concerns in the transformation processes, and C3 classes complement the main key concepts of the domain. The remaining classes leave room for optimization and refinements. This iterative process ensures that the concepts are handled by their increasing importance in the domain and contributes to the final goal of identifying a valuable and efficient property graph.

4.1.4 Application of the patterns and model refinement

This step consists of the application of the patterns to the Class Diagram, according to the tasks defined in the previous step, in order to obtain the first version of the LPG. This preliminary version helps to get an idea of how the data will be represented in the database, and how the queries can be solved. Once the LPG is defined, the next step is to test how suitable it is for answering the queries. Robinson et al. (2015) describes two techniques that can be applied here. The first technique is just to check that the graph reads well. If navigating the graph from a start to an end node we can read off sentences that make sense, the model can be considered reasonably correct. The second technique is to validate that the graph supports the kinds of queries expected to be run on it. If the queries to address the defined use cases can be built (using any graph query language) the graph fits its purpose. This step intends to identify if improvements can be done, as the process is aimed to be as automatic as possible but does not exclude the need for expert validation. This is even more relevant in complex domains or extensive data systems.

4.2 Transformation Patterns

In this work, we are using the main constructs of UML to define the most common situations that could appear when modeling the complexity of any domain. Taking as a basis the UML specification, we have identified basic patterns such as aggregations, compositions, specializations, etc., and we have combined them with the basic possibilities derived from the cardinality options (removing redundancies produced by equivalences between patterns). This allowed us to define the patterns proposed in this section. The defined patterns have been grouped according to their semantics and include an example of application. When the analytical importance of the classes is relevant for the selection of the most adequate pattern (C1, C2, or C3 classes), the classes are highlighted with the “circled A” symbol. This symbol does not belong to the UML class diagram standard and is used in this paper to express this additional property of the class.

The patterns next proposed were specifically defined to address the transformation process proposed in this paper and benefit from considering the analytical context of the domain and the associated requirements, as these are relevant to the identification of LPGs. This analytical context has already shown to be useful in modeling big data warehouses as highly performant analytical systems (Galvão et al. (2020)), with the entities of the domain being classified as entities with high analytical value, entities with high cardinality, and entities with frequent access patterns. In the work of Galvão et al. (2020), three modeling rules allow this classification and, afterwards, the defined patterns allow the identification of the elements available in a big data warehouse, such as analytical objects, complementary analytical objects, special objects, materialised objects, and their corresponding analytical or descriptive families (Santos and Costa (2020)). Starting by the identification of the analytical classes (subsection 4.1.2), the identification of the most adequate patterns (subsection 4.1.3) follows an iterative process that contributes to the identification of a valuable LPG by the application of the transformation patterns.

The transformation patterns proposed here have a strong conceptual background. This means that it is important to determine how expressiveness, at the different levels of abstraction, must be managed. At the "problem space" level (a UML Class Diagram in our case), we identify conceptual modeling patterns that describe what modeling construct must be used to specify the problem under analysis. At the "solution space" level (an LPG in our case), we define the lower-level solution that better represents the concepts with the expressiveness that the technological solution that we use provides. This constitutes the core of a future conceptual model compiler.

When moving from different levels of abstraction, it is common that different conceptual representations can result in the same data structure. This is because the conceptual representation is intended to provide context to the data and ensure the correct understanding of the domain knowledge, while the platform-specific representations are intended to ensure data integrity and meet the analytical requirements. The transformation approach presented in this work tries to merge the advantages of both perspectives allowing the correct evolution and adaptability of the system to the changes. The groups of patterns defined in this work are: Class Patterns, Navigability Patterns, Cardinality Patterns, Generalization Patterns, and Special Patterns.

4.2.1 Class Patterns (CP)

Classes and nodes are used to describe the entities of a domain that share the same features, constraints, and semantics (meaning). The class pattern must be used when a direct equivalence between a class and a node is required. For example, when representing classes with analytical importance as the ones identified as C1, C2, and C3. In this case, the class is represented by a node, the name of the class is the label of the node, and the attributes of the class are the properties of the node (Fig. 2).

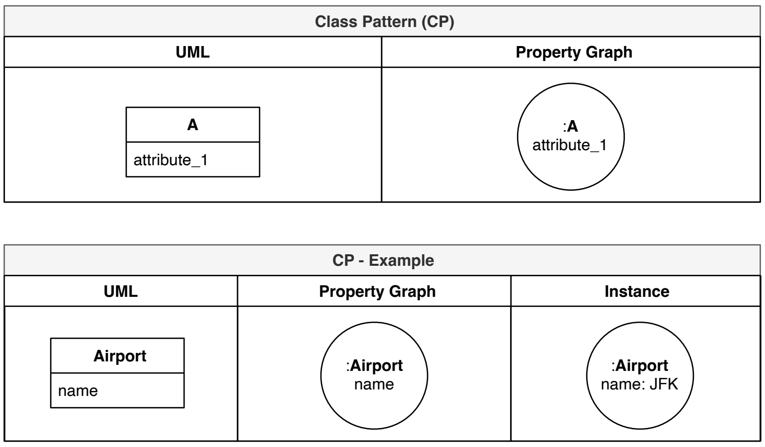

Figure 2: Description and example of the class pattern (CP).

4.2.2 Navigability Patterns (NP)

Navigability represents how the entities of a domain can be accessed from other entities along their associations. Because of the different possibilities to describe the navigability in UML Class Diagrams, the structure could have different representations in the corresponding property graph. According to this, three different navigability patterns have been identified, represented by the associations between classes and the relationships between nodes: NP1 (Fig. 3), NP2 (Fig. 4), and NP3 (Fig. 5).

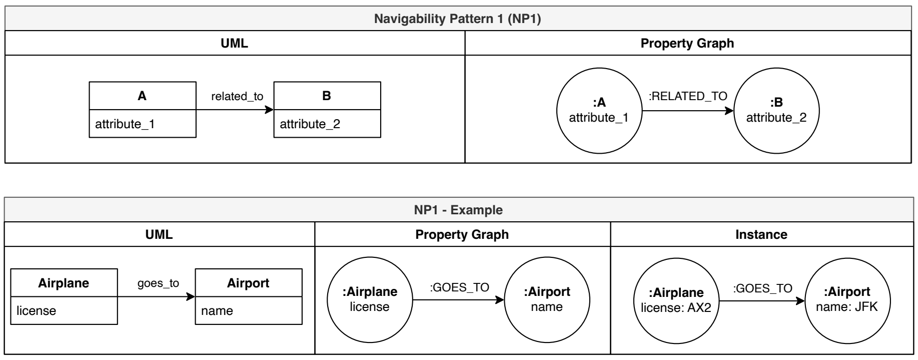

Figure 3: Description and example of the NP1.

The NP1 pattern represents a directed association from class A to class B. This is transformed as a directed graph relationship from node A to node B. In the NP2 pattern, both representations can be considered equivalent even though semantically they are not. In the first case, there is a bidirectional association whose navigability has been explicitly specified. In the second case, the navigability is not specified which means that could be any direction or both. It is a common practice to use this second representation as equivalent to the first. This is why we have considered both options in the same pattern. Nevertheless, in the property graph, the direction of the relationship must be defined, even though it has no implications on the graph database, where nodes can be accessed from any other node by default.

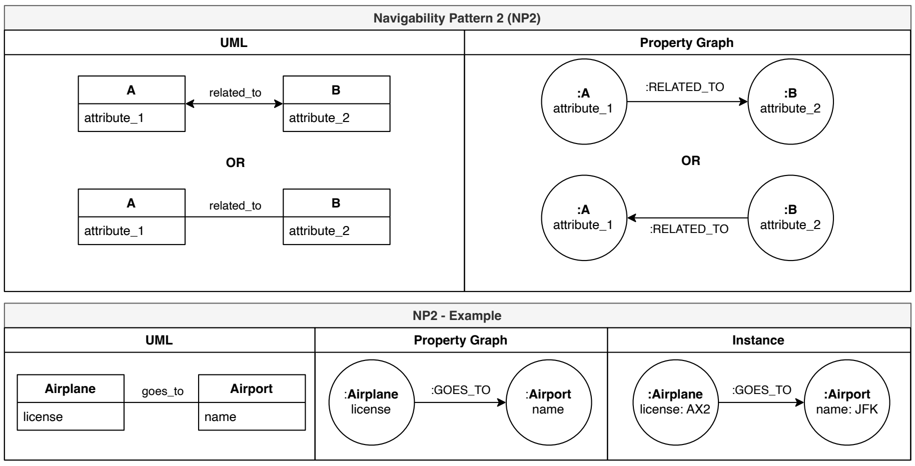

Figure 4: Description and example of the NP2.

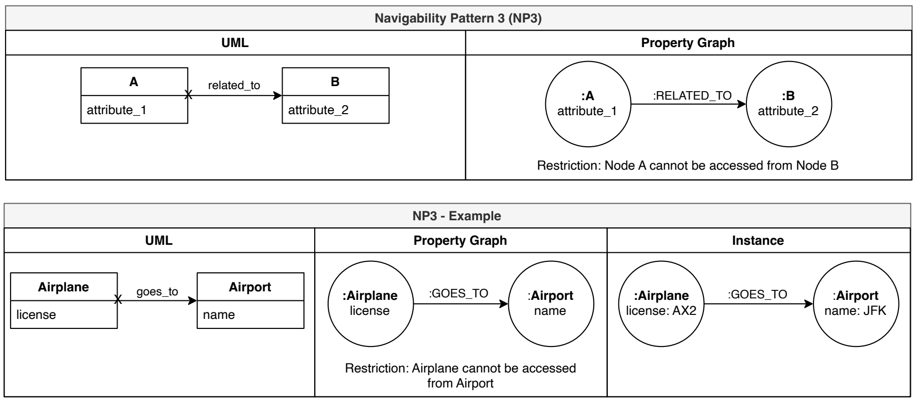

Figure 5: Description and example of the NP3.

Some types of navigation represent intrinsic restrictions that the database must implement to ensure data integrity. For example, NP3 defines that class B can be accessed from class A, but class A cannot be accessed from class B (Fig. 5). This restriction must be internally implemented by the system and is represented as a note in the property graph.

4.2.3 Cardinality Patterns (CdP)

Cardinality is a definition of an inclusive interval of non-negative integers to specify the allowable number of instances of a described element (attributes of a class, or associations between classes). Common multiplicity bounds are zero or more instances (*), at least one instance (1..*), exactly one instance (1), and at least *m* but no more than *n* instances (m..n). In this work, we are focusing on describing the cardinalities of the associations between classes. We also include in this subsection the n-ary associations. In the CdP1 pattern (Fig. 6), each instance of A can be associated with zero or more instances of B, and vice versa.

To simplify the representation, from now on the UML schemas will not consider the direction of the navigability. Since it is mandatory to define the direction in a property graph, it will be represented from left to right by default. Considering that the default cardinality of a relationship in a property graph is "zero to many", the result is equivalent to NP2. In the example, different instances of an Airplane can be associated with zero or more instances of an Airport and vice versa.

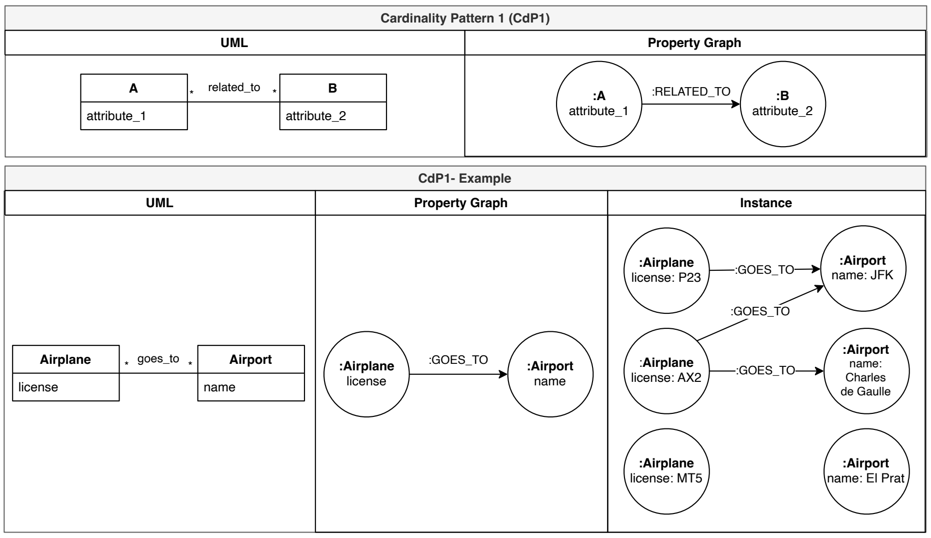

Figure 6: Description and example of the CdP1.

In the CdP2 pattern (Fig. 7), an instance of A must be related to at least one instance of B. Nevertheless, the property graph does not provide any representation of the "one to many" cardinality. In the example, we can see how there are no instances of the Airplane node that are not associated with at least one instance of the Airport node. However, there can be instances of the Airport node that are not associated with any instance of the Airplane node. The application that uses the graph database is responsible for implementing the constraints to ensure consistency according to the UML definition.
The CdP3 pattern (Fig. 8) represents a more restrictive case of the CdP2 pattern. In this case, an instance of A must be associated with one - and only one - instance of B (CdP3-A), or can be associated at most with one instance of B (CdP3-B). The representation in the property graph does not allow the possibility of representing these cardinalities, thus the resulting model is the same as CdP2. The application that uses the database should implement the corresponding constraints. When only one class has analytical relevance (CdP3-C), the resulting property graph merges the classes in the node that has the analytical relevance. When the cardinality allows only one instance of each class, their analytical relevance is important as can be seen in Fig. 9. The CdP4-A pattern is useful if both classes have, or do not have, analytical relevance. When only one class has analytical relevance (CdP4-B), the resulting property graph merges the classes in the node that has the analytical relevance.

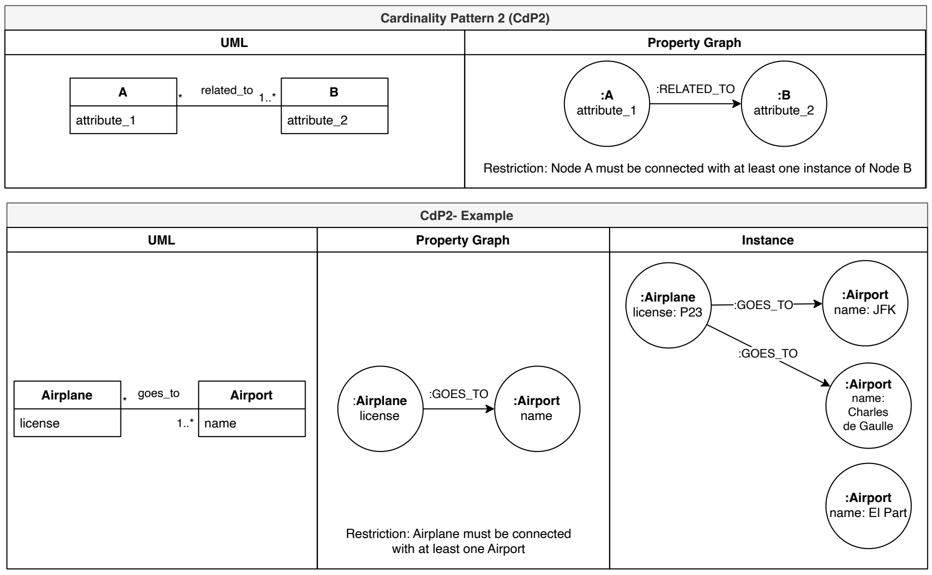

Figure 7: Description and example of the CdP2.

An n-ary association relates three or more classes (CdP5). The multiplicity of n-ary associations defines how the relationships between classes are represented in the property graph (Fig. 10).

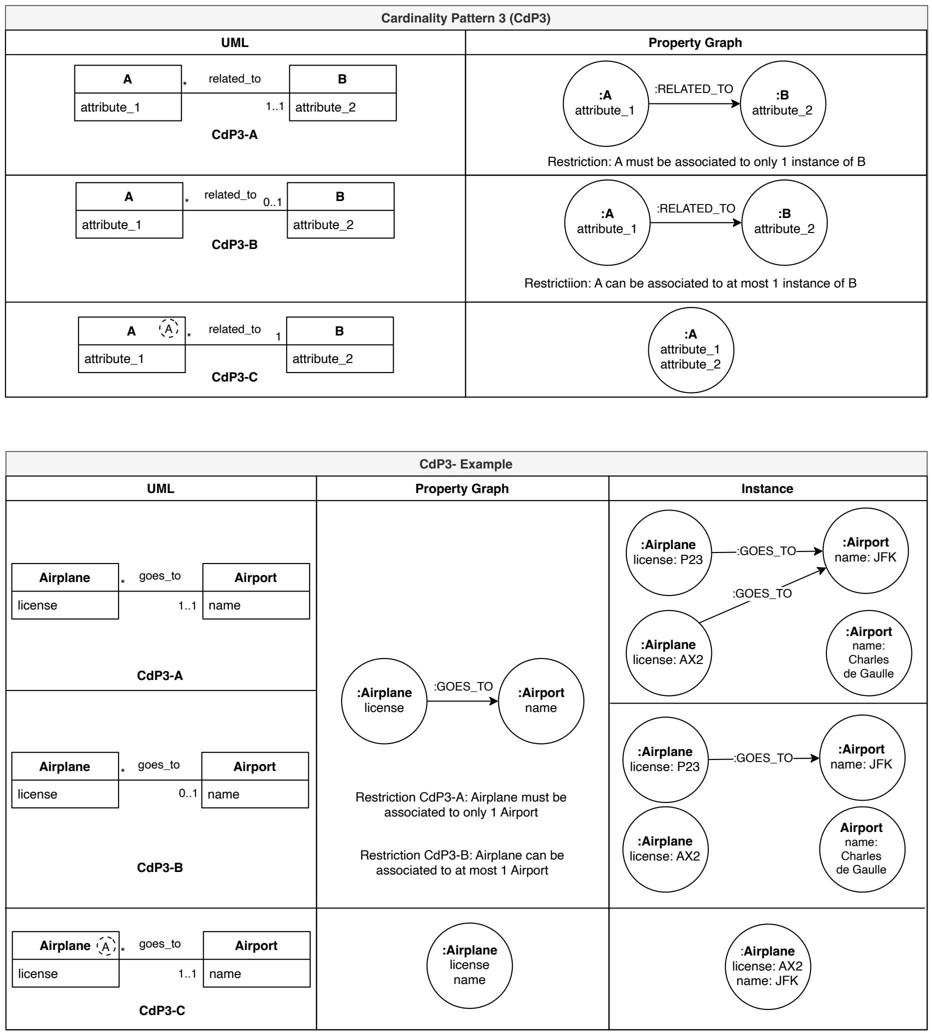

Figure 8: Description and example of the CdP3.

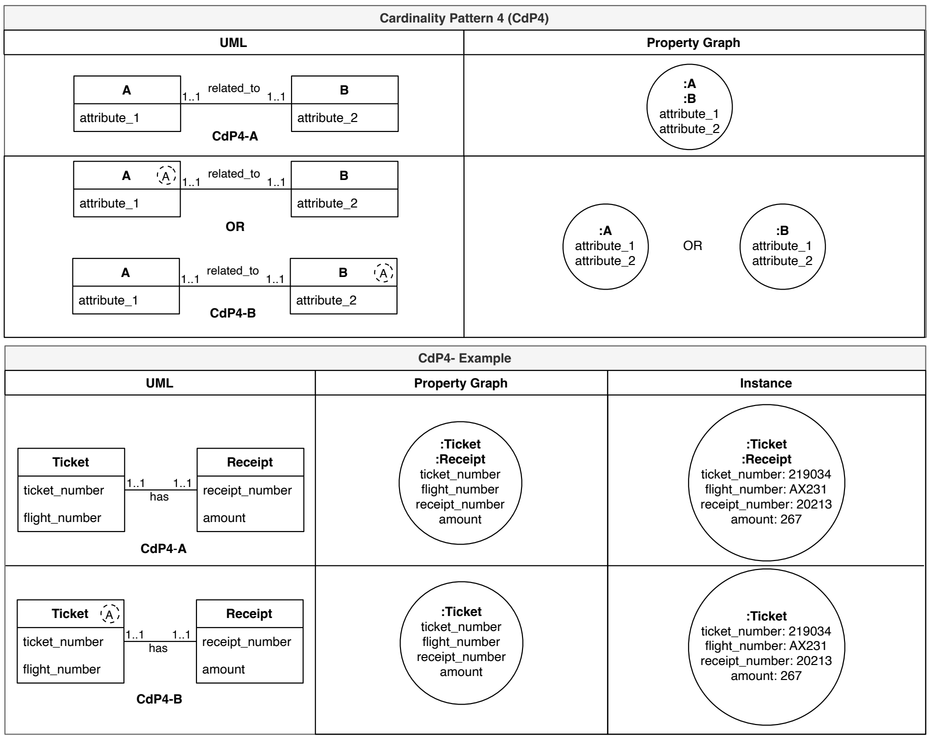

Figure 9: Description and example of the CdP4.

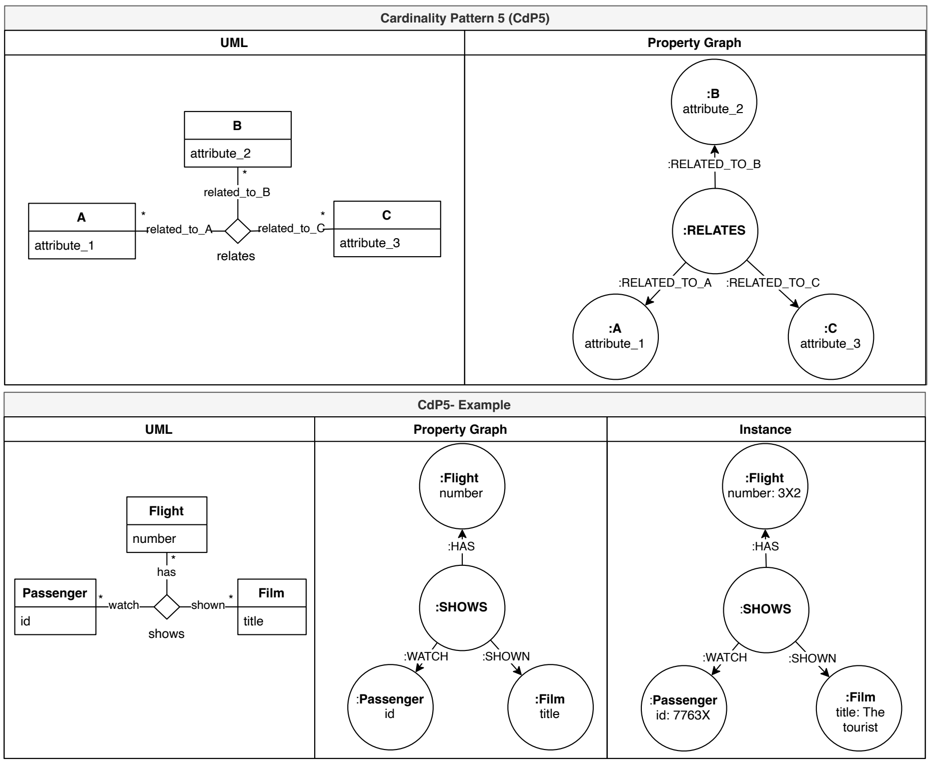

Figure 10: Description and example of the CdP5.

4.2.4 Generalization Patterns (GP)

Generalization is a binary association between a more general classifier (superclass) and a more specific classifier (subclass). Generalizations can have 4 different constraints: i) Incomplete: there could be instances of the general classifier that could not be classified as any of the specific classifiers; ii) Complete: every instance of the general classifier is also at least one instance of the specific classifiers; iii) Disjoint: no instance of any specific classifier may also be an instance of another specific classifier; and, iv) Overlapping: an instance of the general classifier may also be an instance of more than one of the specific classifiers. These constraints are usually represented as pairs (e.g., complete/disjoint, incomplete/overlapping). The incomplete/disjoint pair means that superclass A cannot be specialized into subclasses A1 and A2 at the same time. The incomplete/overlapping pair means that the superclass A can be specialized into the subclasses A1 and A2 at the same time.

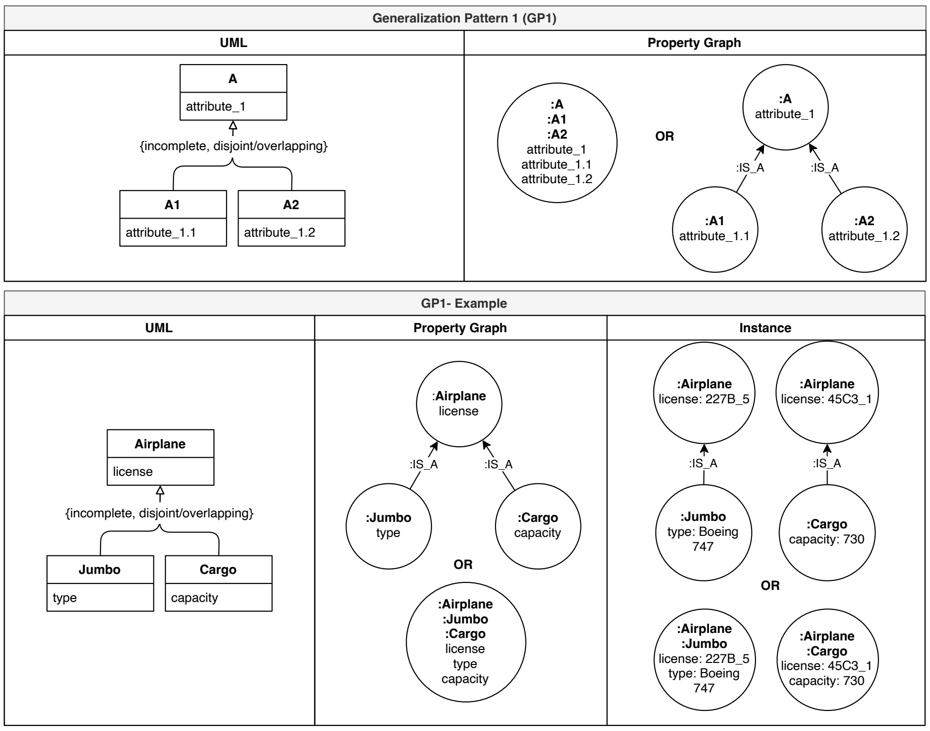

Figure 11: Description and example of the GP1.

The incomplete restriction (Fig. 11) leads to the same pattern independently of the disjoint or overlapping restriction. The complete/overlapping pair (Fig. 12) means that the superclass must be specialized into one or both subclasses. And, finally, the complete/disjoint pair represented in Fig. 13 means that the superclass A must be specialized into the subclass A1 or the subclass A2, but not both.

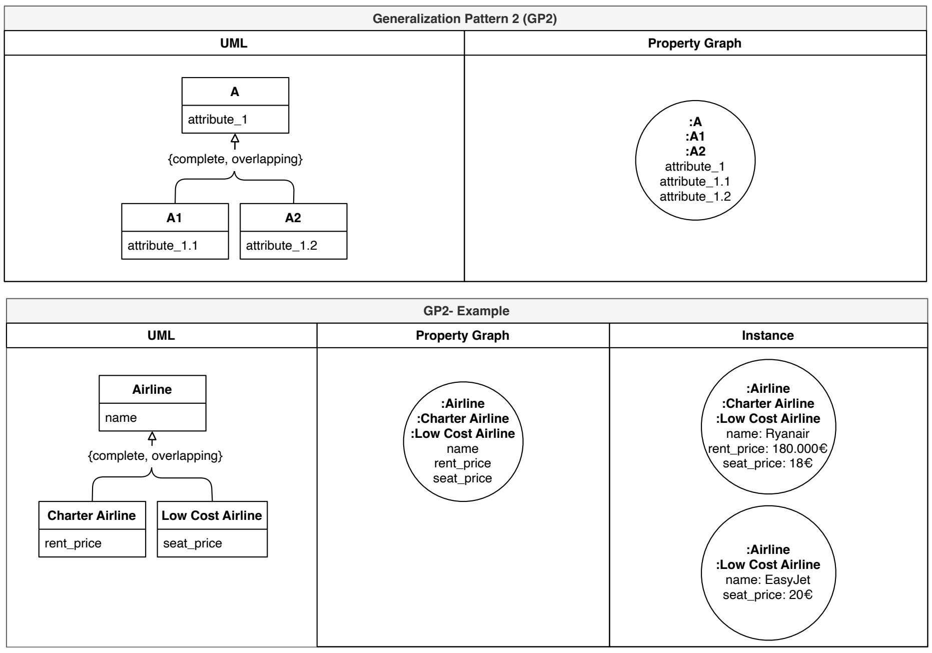

Figure 12: Description and example of the GP2.

4.2.5 Special Patterns (SP)

This subsection includes special UML representations such as aggregations, compositions, and association classes. The aggregation is a binary association that represents a whole/part relationship (SP1) as can be seen in Fig. 14. The composition is a kind of association where the composite object has sole responsibility for the existence and storage of the composed objects (SP2). As can be seen in Fig. 15, two cases are identified, depending on the analytical relevance of the classes. When implementing a composition, a restriction must be also implemented to ensure that if the whole class is removed the part(s) must also be removed (SP2-A).

Finally, the association classes are used to add attributes, operations, and other features to associations (SP3) as represented in Fig. 16. The first case of the SP3 is used when the association class is not associated with any other class and does not have analytical importance. This means that the attributes can be represented as properties of the relationship between the other nodes (SP3-A). If the association class is associated with any other class (SP3-B) or has analytical importance (SP3-C), it must be represented as a node to maintain the relationship with other nodes.

5 Practical Example: Flight Management System

This section describes how the transformation rules and patterns can be applied to a practical example: a flight management system. A flight is defined by a number and a date and corresponds to a route

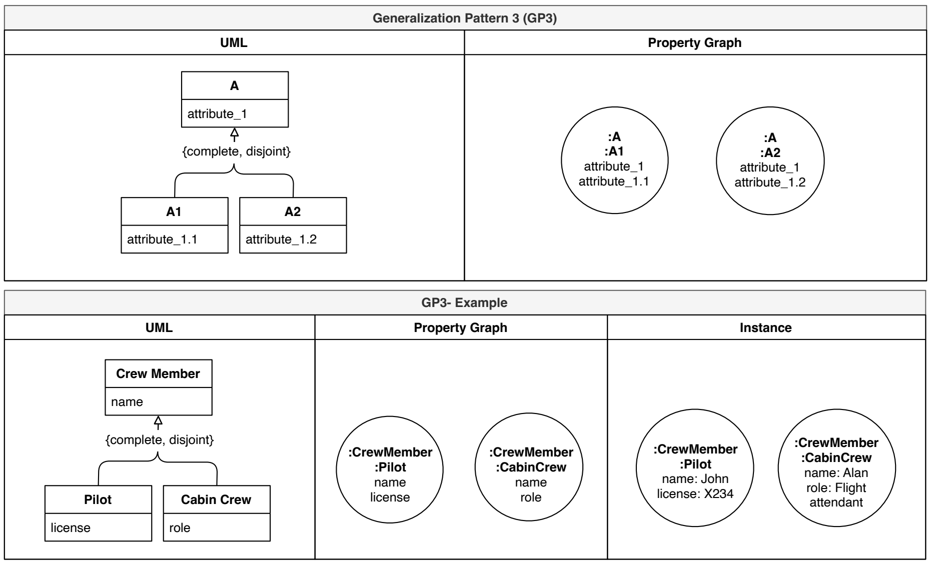

Figure 13: Description and example of the GP3.

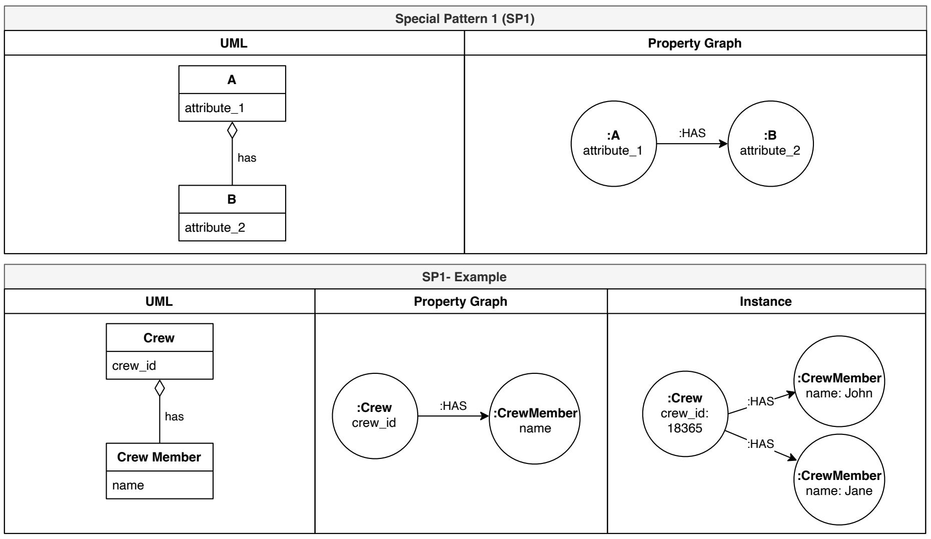

Figure 14: Description and example of the SP1.

from an origin airport to a destination airport. There may be several flights that share the same origin and destination airports. An airport has a name and is located in a city of a country. An airplane is assigned to a flight and it is defined by a license number. Airports have a number of baggage belts where a customer can collect the baggage when the airplane arrives at the destination airport. Customers are described by their passport number, name, and surname. They make reservations for specific seats on specific flights. A flight has a crew, composed of a pilot, a co-pilot, and a varying number of cabin crew members. Crew members are identified by a crew id, name, and surname. Pilots and co-pilots are defined by their license number and the number of flight hours. The cabin crew is also identified by the role they have. Crew members work for an airline. A flight is operated by an airline. An airline can operate more than one flight and is defined by its name. The airline is in charge of registering the operation status of a flight.

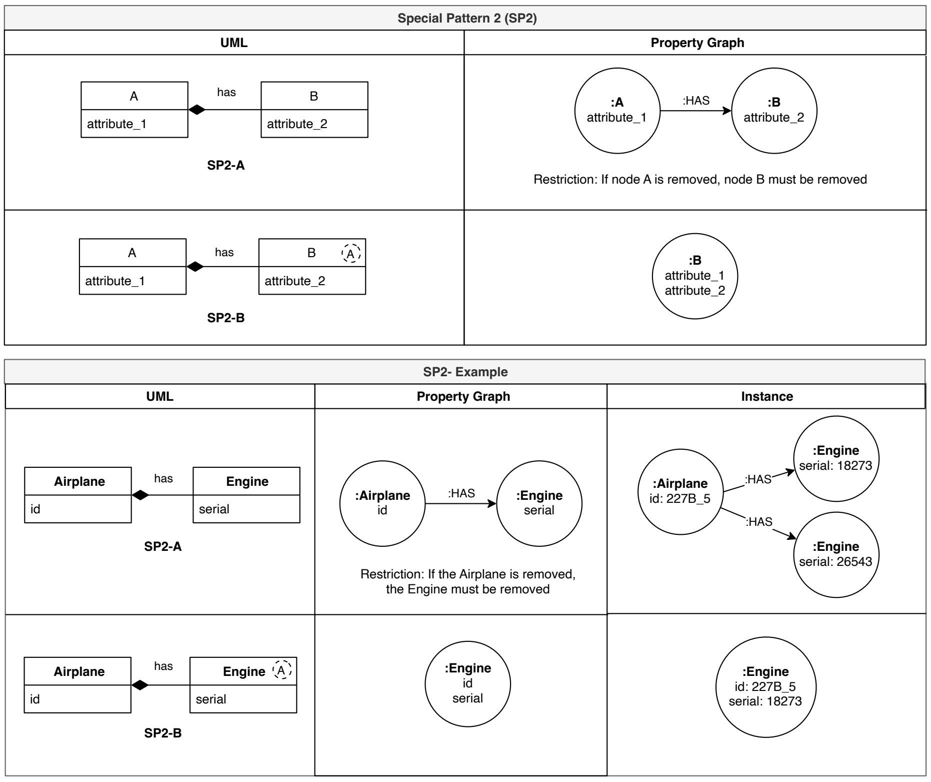

Figure 15: Description and example of the SP2.

This register includes the date, the hour, and the status (boarding, closed, departure, arrival, etc.).

5.1 Identification of the Analytical Requirements and the Analytical Classes

Following the steps defined for the transformation rules, we start with the definition of the analytical requirements, expressed by the queries to be answered. In this example, we act as experts in the domain, and based on our experience we have defined some examples of interesting queries:

- Q1: Identify the **flights** to specific **airports**.
- Q2: Identify **flights** that go to the same **airport**.
- Q3: Find **flights** that depart at some **date** at a specific **airport** and go to a given **location** on a specific **date**.
- Q4: Identify the **flights** performed by a specific **crew member**.
- Q5: Identify which **airplanes** usually perform a specific **route**.
- Q6: Find the **baggage belt** of a specified **flight**;
- Q7: Identify the rate of delayed **flights** (**status** = delayed) in a specific **route**.

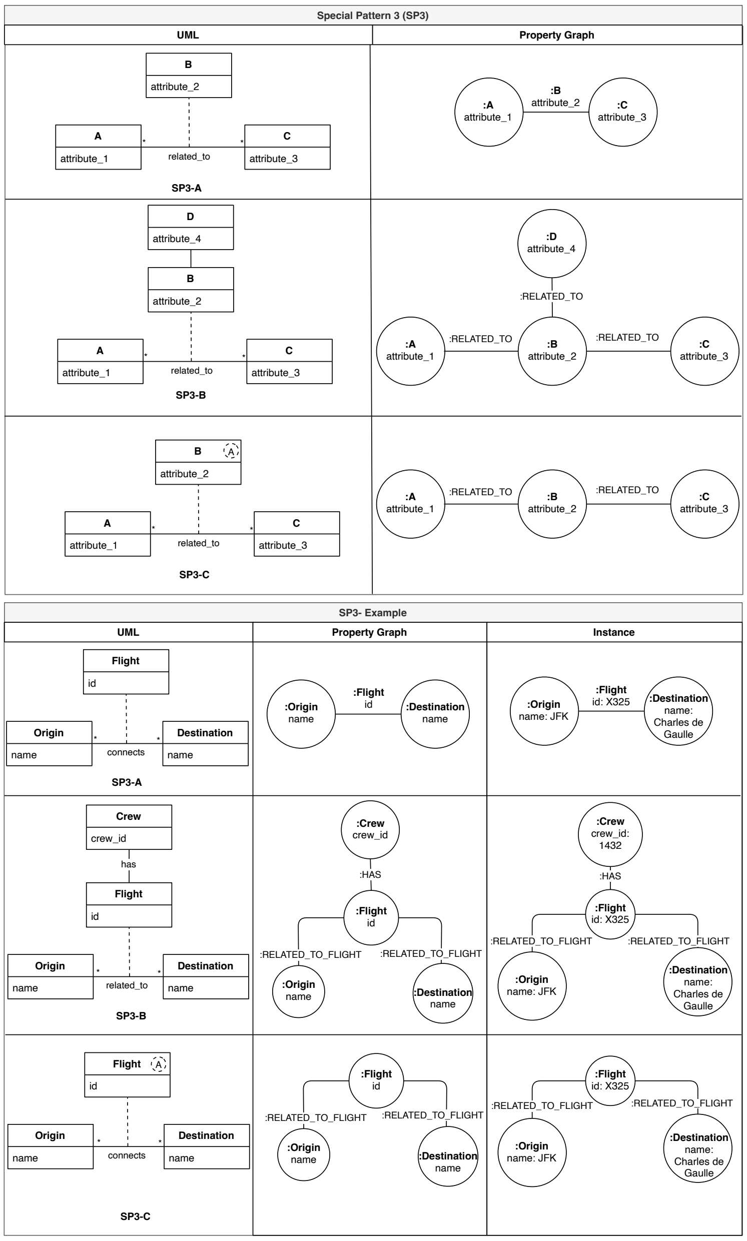

Figure 16: Description and example of the SP3.

- Q8: Identify the **airlines** with the higher rate of delayed **flights** (**status** = delayed).
- Q9: Identify the **customer** of each seat in a specific **flight**.
- Q10: Find the **crew members** of a specific **flight**.

In these queries, the key concepts have been highlighted in bold. In summary, we can conclude that the most relevant concepts for analytics are flights and airports because they appear in most queries. Additional interesting concepts are location, crew member, airline, airplane, baggage belt, customer, seat, status, and date. The route is used as a synonym for flight. Using the description of the domain and the above-described queries, we have defined the corresponding UML Class Diagram depicted in Fig.

17.

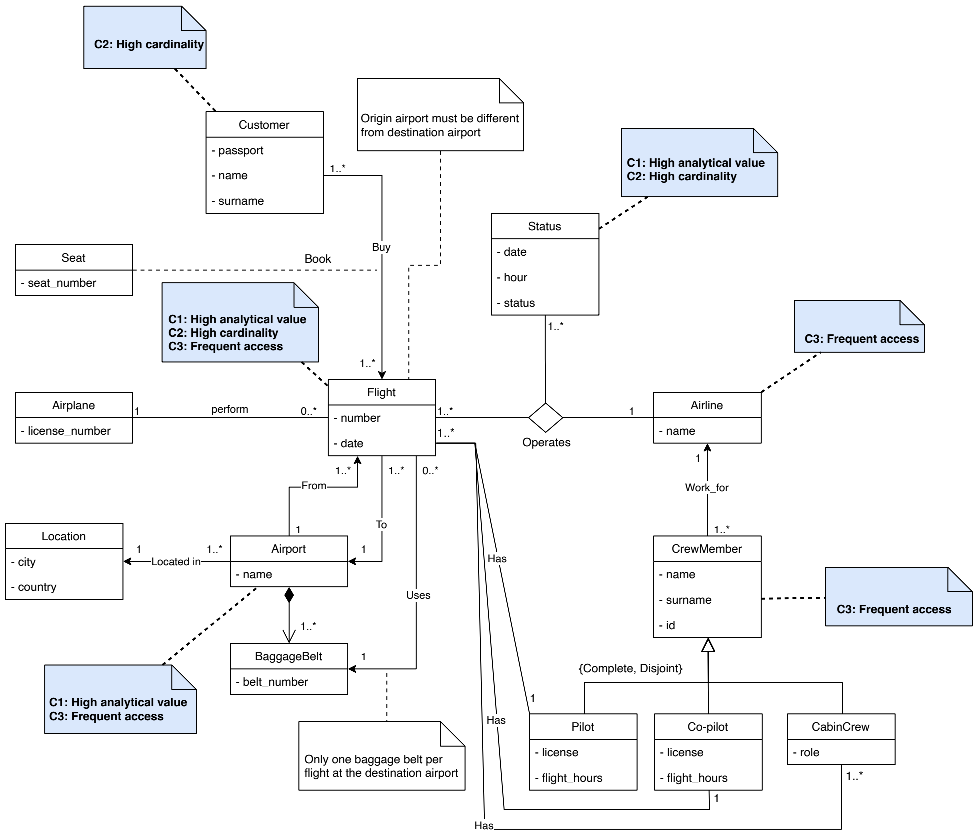

Figure 17: Analytical classes identified according to the analytical requirements.

According to the requirements, the following classes have been identified as having analytical interests:

- Flight and Airport are the classes with the higher analytical value (C1) and are present (one of them, at least) in all the defined queries (C3). The number of airports is not expected to be high, but the number of flights managed by the system is expected to be much larger (C2).

- Crew Member is a class used to answer two queries that aggregate flights by a crew member and vice versa (C3). Nevertheless, it is not expected to have a huge volume of crew members but it has analytical importance like other classes such as Airport and Flight have. The specializations (Pilot, Co-pilot, and CabinCrew) are not considered to have analytical value because they do not participate in any query.

- Status is expected to have a high volume of instances (C2) and a high analytical value (C1) since there are two queries that aggregate flights and airlines by status. Moreover, from a management perspective (e.g., airline or customer) it could be relevant to have performance indicators to measure how well, or not, an airline or route is performing.

- The Airline is not expected to have a high volume of instances, nevertheless, it is frequently accessed (C3) to answer queries that relate to Flight and CrewMember, or to address the performance indicators previously mentioned.
- Customer is a class that is expected to have a high volume of instances (C2).
- Airplane, Location, Seat, and Baggage Belt are not expected to have a high number of instances, they only appear in one query, and they are not one of the main concepts of the domain. Therefore, they are not considered to have analytical importance.

All these classifications are dependent on the analytical context and, therefore, guide the transformation process to a property graph that efficiently represents the application domain.

5.2 Identification and Application of the Most Adequate Patterns

In this subsection, we explain how the different patterns were applied according to the analytical relevance of each class. The first step requires to represent C1 classes as nodes and their relationships with other C1 classes. To highlight their importance in the application domain, Flight, Status, and Airport classes will be represented as nodes (pattern CP1). As these three classes are directly related in the UML Class Diagram, the navigability patterns must be considered (in this case pattern NP1). Any constraint must also be specifically described (Fig. 18).

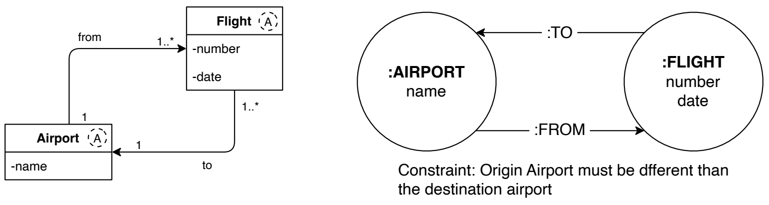

Figure 18: Transformation of the relationships between Airport and Flight.

The Status class belongs to a tertiary relationship (with Flight and Airline), so the applied pattern is CdP5 (Airline has not been defined yet so it is temporarily represented as a grey node) (Fig. 19).

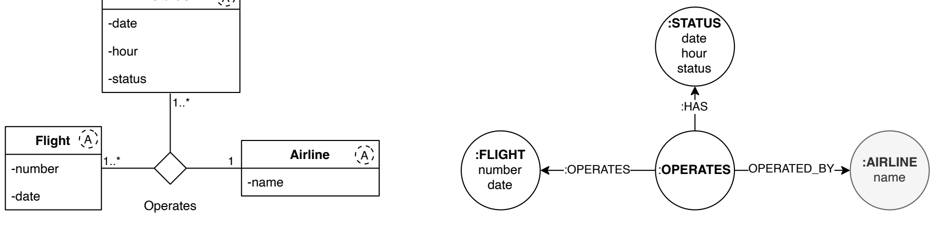

Figure 19: Transformation of the relationships between Status and Flight.

The second step requires representing C2 classes as nodes and their relationships with C1 and C2 classes. In this example, Customer is represented by a node (pattern CP1) with specified navigability in the association with Flight (pattern NP1) (Fig. 20).

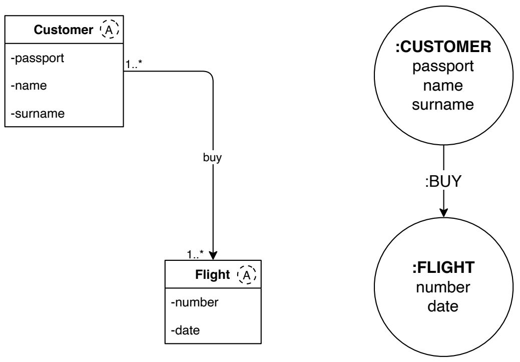

Figure 20: Transformation of the relationships between Customer and Flight.

In the third step, the C3 classes are represented as nodes, establishing their relationships with C1, C2 and other C3 classes. In this example, Airline is a frequent access class. Therefore, it is represented as a node and takes part in the tertiary relationship with Status and Flight (pattern CdP5). The NP1 pattern is used to represent the binary association between CrewMember and Airline and the associations between Flight and the CrewMember specializations (Fig. 21). The CrewMember class includes a specialization therefore, the pattern used is GP3 (Fig. 22). Flight and Airport, also classified as C3, are not explained here since they have been already identified in previous steps.

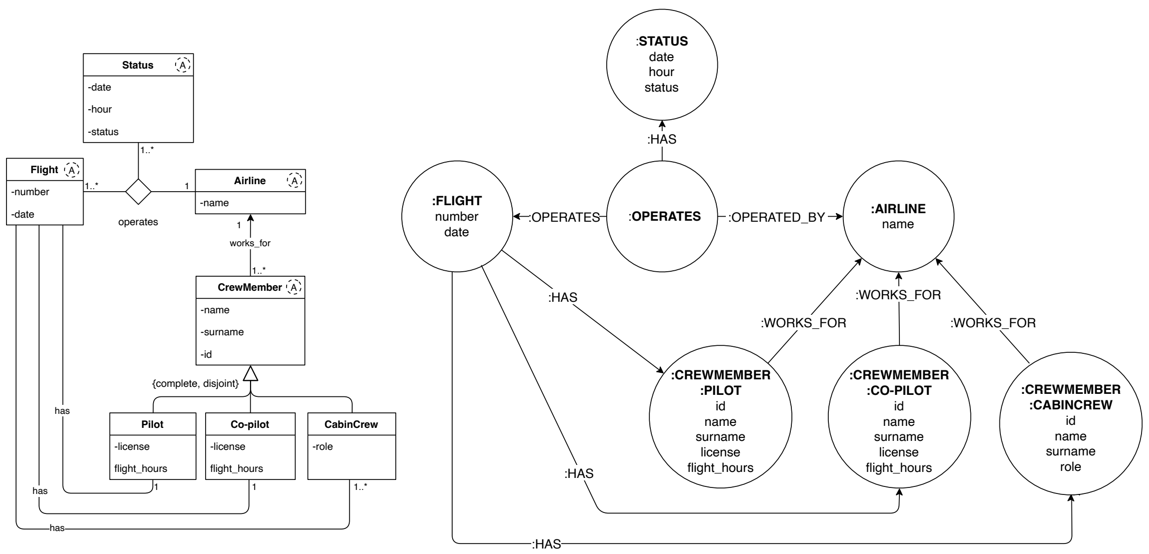

Figure 21: Transformation of the relationships between Airline, Status, Flight, and CrewMember.

Finally, step 4 requires to represent all the remaining classes and their associations with other classes. In this example, the classes that do not have analytical relevance are Seat, Location, Airplane, and Baggage-

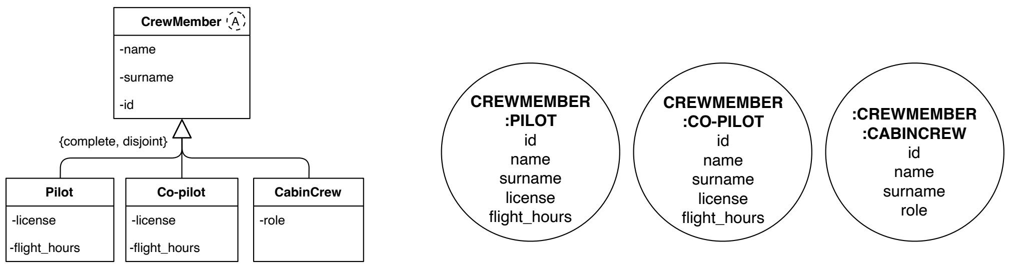

Figure 22: Transformation of the relationships between CrewMember, Pilot, Co-Pilot, and CabinCrew.

Belt. Seat is represented as an association class. Since it does not have associations with any other class, the pattern selected is SP3-A. The attribute of the Seat class (seat_number) is added to the association previously created between Customer and Flight (Fig. 23).

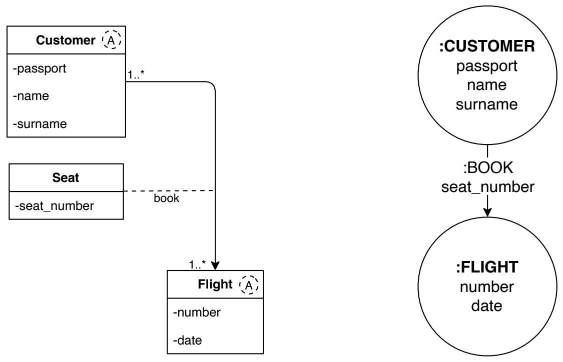

Figure 23: Transformation of the relationships between Seat, Customer, and Flight.

According to the multiplicity of the association between Location and Airport, the pattern selected is CdP4-B. The attributes of Location (city and country) are added as properties to the Airport node. A constraint has been defined to ensure that the location is always defined (either by the city or the country) (Fig. 24). For Airplane, according to the multiplicity of the association between Airplane and Flight, and the fact that Flight is the class with analytical value, the pattern selected is CdP3-C (Fig. 25).

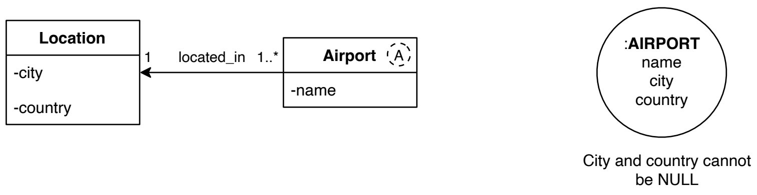

Figure 24: Transformation of the relationships between Location and Airport.

The BaggageBelt class is associated with Airport by composition, and with Flight by a binary association. The patterns selected are SP2-A (between Airport and BaggageBelt), NP1 (between Flight and BaggageBelt), and CdP3-A (between Flight and BaggageBelt). The corresponding constraints have been also added to the property graph (Fig. 26).

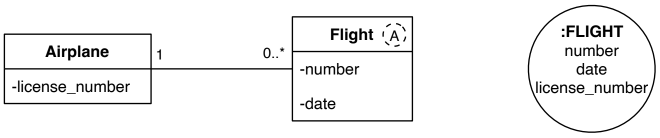

Figure 25: Transformation of the relationships between Airplane and Flight.

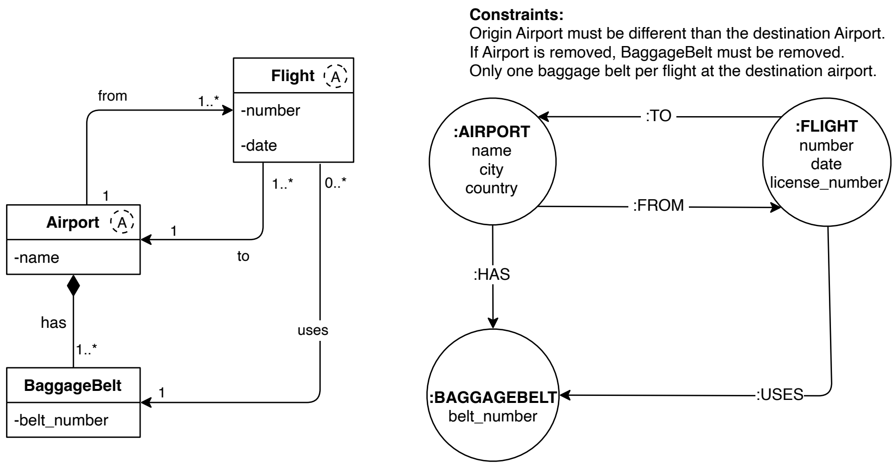

Figure 26: Transformation of the relationships between Baggage belt, Airport, and Flight.

After the selection of the most adequate patterns, the resulting property graph is shown in Fig. 27.

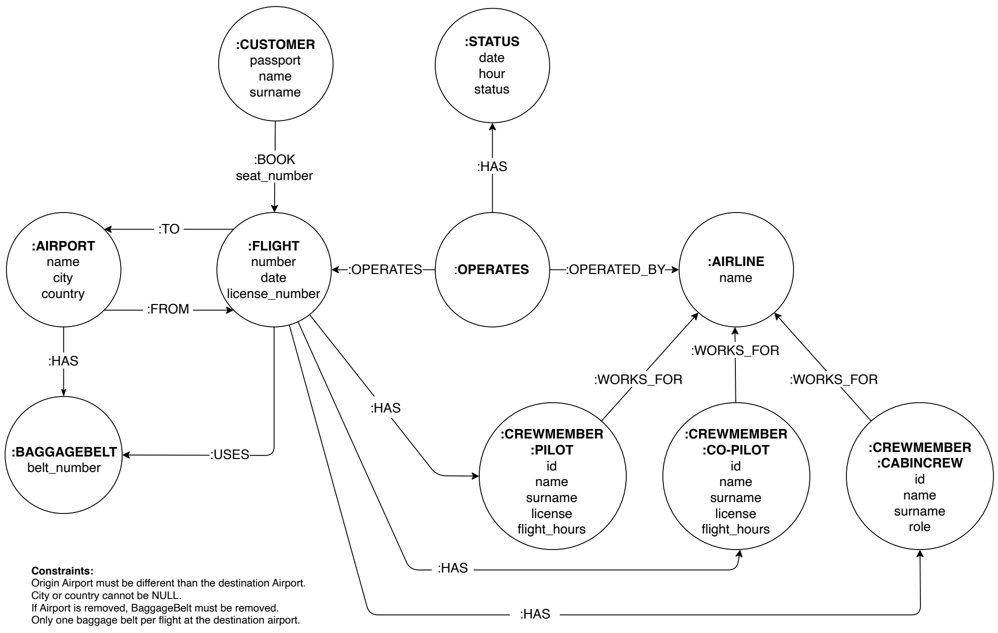

Figure 27: Resulting property graph.

The last step of the process is model refinement, where the user checks the resulting graph in case additional transformations are required. In this case, the resulting graph is adequate to answer the queries and we do not consider additional modifications are required.

6 Evaluation

This section evaluates the differences between the application of a classical approach and the approach proposed in this work. By classical approach, we mean the direct equivalence between nodes and associations and classes and relationships done by the already available solutions. The evaluation is based on the assessment of different dimensions to i) prove that the resulting LPG is semantically equivalent to the original model and still preserves its original behavior, and ii) verify that the proposed approach highlights certain model characteristics that enhance the overall quality of the model.

To verify that the model obtained using the approach proposed in this work is equivalent to the original UML model, we measure the semantic equivalence between both models. To verify that the proposed approach improves the quality of the resulting model, we compare the readability, maintainability, complexity, size, and performance of two LPGs; one LPG obtained using the classical approach (Fig. 28) and another LPG obtained using the proposed approach (Fig. 27). Both models are obtained using the UML Class Diagram used as an example in Section 5.

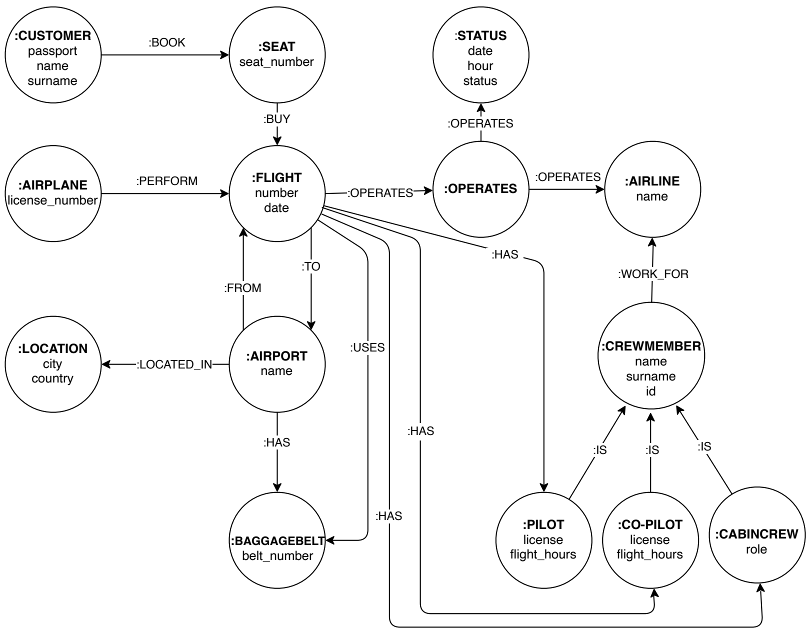

Figure 28: Labeled Property Graph obtained using the classical approach.

The LPG shown in Fig. 28 has been obtained using a direct transformation where a node represents a UML class, and each UML association is represented using a relationship between nodes.

6.1 Semantic Equivalence

After transforming the UML model, proving that the resulting LPG is semantically equivalent to the original model, and still preserves its original behavior, is crucial. In general, for two models to be semantically equivalent, they must have the same observable output when executed under identical inputs. Therefore, in this example, we measure the semantic equivalence of both models (the original UML and the resulting LPG) by performing the same queries and verifying that they can be solved. The queries to be solved are extracted from the analytical requirements, previously presented in Subsection 5.1.

As shown in Fig. 29, all the queries can be solved in the resulting LPG, navigating through the marked paths to access the relevant nodes and their attributes. This confirms that the resulting LPG complies with the analytical requirements and is semantically equivalent to the UML Class Diagram.

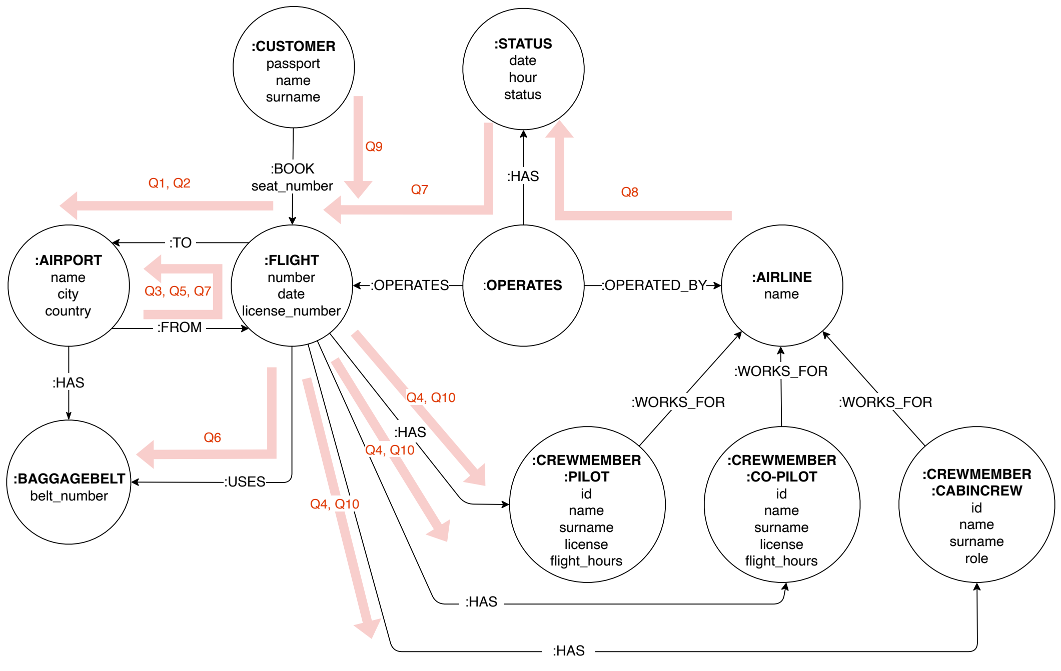

Figure 29: Paths to solve the queries that satisfy the analytical requirements.

6.2 Model Quality Comparison

The classical transformation approach makes the direct correspondence between classes and nodes, and between associations and relationships. This means that this classical approach does not consider the most efficient transformation according to the analytical value of each class. Therefore, the resulting property graph would have "less" quality to fulfill the analytical requirements. To demonstrate that the proposed approach produces a model with higher quality and more aligned with the analytical requirements, we measure different metrics over both models and compare the results.

6.2.1 Size and Readability

The size of an LPG can be expressed in terms of the number of nodes or the number of relationships, as these are the two main components of the graph. Some LPGs may have a high number of nodes, which will naturally lead to a high number of relationships, increasing the complexity of the queries to achieve a certain knowledge requirement, the clarity of the resulting model, and its maintainability. The size of the model also impacts its readability, commonly described as the degree to which a schema represents the modeled domain in a natural and clear way, with the aim of being self-explanatory to the user (Ehrlinger, Huszar, and Wöß (2019)).

In this example, the model obtained using the proposed approach has four nodes and four relationships less than the model obtained using the classical approach, corresponding to a decrease of 28.5% and 22%, respectively. This will impact the readability and maintainability of the model.

6.2.2 Maintainability

When a change is performed on one part of the system, it may impact the way the analytical requirements are fulfilled (e.g., when executing the queries to extract the required knowledge). The maintainability of a model can be measured by estimating the average change impact of any potential change applied to the model (Almasri, Korel, and Tahat (2017)). This measure gives a better understanding of how easy or difficult the maintenance of the model is. The smaller the average change impact is, the easier it is to maintain the model.

An approach to identifying the impact of a change in an LPG can be measured by estimating the average number of queries affected by changes in any relationship of the model. These changes can be adding, removing, or editing relationships. Although we mainly focus on modifications applied to relationships, modifications applied to the nodes are inherently considered in our approach. For example, when a new node is added, either new relationships are added to link the new node to other nodes in the model, or existing relationships will change their originating/terminating node to connect the new node to other nodes in the model. Considering this approach, we have measured the number of changes that may affect each query as the number of relationships involved in solving it. For example, to solve query 1 in the graph shown in Fig. 27, the Flight and Airport nodes, and the TO relationship, are involved. Any change in this relationship will affect the query and, therefore, the change impact, in this case, is 1. The higher number of relationships required to solve a query, the more prone to change impact the query is and, therefore, the more complex to maintain.

Comparing both models (Table 1), we see how the average change impact has decreased from 2.4 to 1.9 (20.8%). This means that the proposed model is more ease to maintain than the one obtained following the classical approach. Although for three queries there is an increase of the change impact, globally the current approach maintains (for three queries) or decreases (for four queries) the change impact. The proposed approach decreases the transition density by 50% in Q9 and Q10 and by 33.3% in Q3 and Q5.

	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**	**Average Impact**
Classical	1	1	3	1	3	1	3	1	2	6	**2.4**
Proposed	1	1	2	2	2	1	4	2	1	3	**1.9**

Table 1: Comparison of the average change impact between both LPGs.

6.2.3 Complexity

The complexity of an LPG can be measured by estimating the model's transition density. The model's transition density was first defined by Almasri, Tahat, and Korel (2022) as the average number of transitions per state (num_ransitions/num_states) in an Extended Finite State Machine (EFSM). Considering the path to solve a specific query as a EFSM where the nodes and relationships establish the route to get the required result, we can approximate the complexity of an LPG through the calculation of the query complexity of each of the queries that are required to fulfill the analytical requirements. For example, in the proposed approach, two nodes and one relationship are required to solve query 1. Therefore, the complexity of the portion of the model required to solve this query is 0.5.

Comparing both approaches (classical and proposed), Table 2 shows how the overall transition density has decreased from 0.771 to 0.692 (10%). This means the proposed model is less complex than the one obtained following the classical approach. Also, it is important to highlight that, according to this metric, no query increases its complexity, maintaining (in 7 queries) or decreasing (in 3 queries) the transition density. In Q10, the proposed approach decreases the transition density by 37,5%.

	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**	**Average Complexity**
Classical	0.5	0.5	1	0.67	1	0.5	1	0.67	0.67	1.2	**0.771**
Proposed	0.5	0.5	1	0.5	1	0.5	1	0.67	0.5	0.75	**0.692**

Table 2: Comparison of the average change impact between both LPGs.

6.2.4 Performance

The performance of an LPG can be measured in terms of the time each query takes to finish. To such an aim, we have executed a benchmark to measure the query performance of both models in three different datasets with increasing sizes and representing the same data. The aim of this benchmark is twofold: i) to ensure that reducing the size of the model does not increase the execution time and ii) to prove that the proposed approach enhances the performance of the queries, especially those involving grouping or long paths. The benchmark has been executed in an Intel Core i7 machine with 8GB of RAM, Windows 10, and Neo4J 5.2. Each query has been executed 100 times and the final query execution time is the average time of the several runs, removing the first run as this is affected by the Cypher query planner. The size of each dataset (number of nodes and relationships) is represented in Table 3.

Dataset Name	**Num. Nodes**	**Num. Relationships**
DS1	1,900	2,050
DS2	19,000	20,500
DS3	190,000	205,000

Table 3: Dataset size (nodes and relationships) used in the benchmark.

According to the queries defined as requirements and the differences between the classical approach and the one proposed in this work, the queries can be divided into two groups: i) queries where the subgraph required to answer the query is the same in both approaches, and ii) queries where the subgraph required to answer the query is smaller in the proposed approach. The queries of the first group are Q1, Q2, Q6, Q7, and Q8. The queries whose associated paths are reduced are Q3, Q4, Q5, Q9, and Q10. As can be seen in Fig. 30, there are no significant differences in the execution times of the queries with the same subgraph (Q1, Q2, Q6, Q7, and Q8), with the proposed approach reducing by a maximal average difference of 1,76% in DS2. Nevertheless, it is worth mentioning that for the queries where the subgraph is shorter, the proposed approach reduces the processing time by a maximal average difference of 15,03% for DS3. As can also be seen in the chart, as the size of the dataset increases, the average time variation also increases, which means that with increasing datasets the proposed approach tends to be even more efficient than the classical one.

Figure 31 provides an overview of the average time variation by query for all datasets, highlighting the time variation between the classical and the proposed approach. As shown in this chart, the most significant average time variation is produced in queries Q3, Q9, and Q10. These are queries where

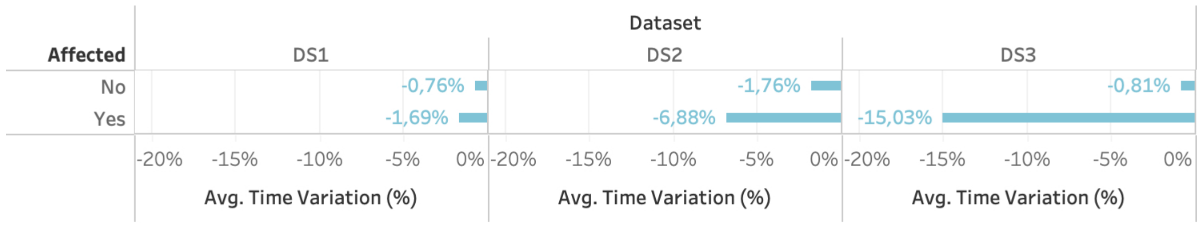

Figure 30: Average time variation between queries whose length subgraph has been affected and those whose length subgraph is the same.

transition density has been reduced by 33% or 50%.

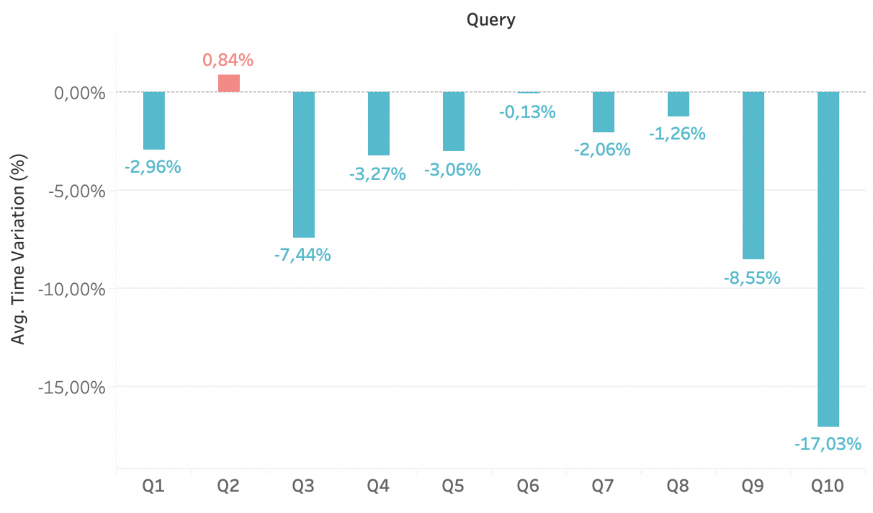

Figure 31: Average time variation by query for all datasets.

An increment of 0.84% in the average time has been observed in Q2. Since the datasets for the classical and proposed approaches contain the same data, and Q2 is the same in both approaches, the difference is the number of attributes of Airport. In the proposed approach, Airport has two more attributes, corresponding to Location. Nevertheless, other nodes such as CrewMember have more attributes too and the corresponding queries are not penalized. Therefore, we conclude that this is not the cause of the difference in time measurement and is so small that does not affect the overall comparison between models.

In most of the queries and workloads, the proposed approach obtains better processing times, especially for DS3, with 190.000 nodes and 205.000 relationships, where the differences between the processing times of the affected queries are more expressive (Fig. 32). This corroborates the hypothesis stated to run this benchmark: considering the analytical requirements, the proposed approach not only produces an LPG that maintains the execution times but also improves the performance of some queries.

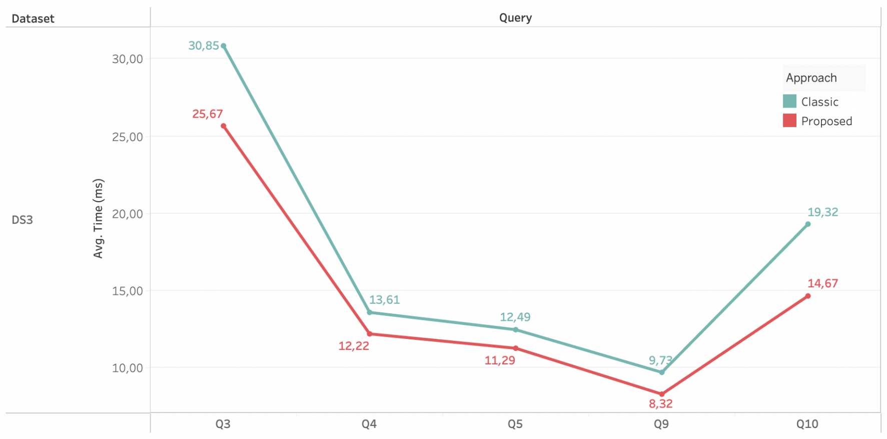

Figure 32: Average time variation in DS3 for queries whose subgraph length is reduced.

7 Conclusion and Future Work

This work proposes a systematic approach to transform conceptual schemas, represented as UML Class Diagrams, into LPGs by using a set of transformation rules, patterns, and steps applied in a systematic way. The approach was applied to a demonstration case and validated by comparing two competing LPGs and measuring different quality dimensions (semantic equivalence, readability, maintainability, complexity, size, and performance). According to the results, the obtained LPG is not only semantically equivalent but also enhances the above-mentioned characteristics which produce a model easier to understand and maintain and more aligned with the analytical requirements. The set of patterns presented covers cases that commonly appear in Class Diagrams. Nevertheless, when moving from different levels of abstraction, multiple transformation options for one pattern may arise. The identification of such situations and the selection of the transformation strategy are important aspects of this approach, which we tried to accomplish by adding the evaluation of the analytical importance of each class. This guides the selection of the most suitable transformation by adding some context that only the user can establish. Nevertheless, the experience of the designer must be considered and the final decision, if more than one solution is available, must rely on him. This is a strategy followed by different transformation tools, that cannot solve conflicts and allow the user to decide which is the best option in these situations.

Due to the importance of the analytical relevance of each component, this set of patterns must be tested and refined by their application in different and complex knowledge domains. Different domains, with the corresponding class diagrams representing the domain's concepts, to be able to cover the whole set of proposed transformation patterns and have different sequences and combinations of the patterns to apply. Complex domains, with highly interconnected data, to ease the adoption of graph approaches for the storage and processing of data whose value derives from the efficient analysis of those relationships, a characteristic of graph-based data systems. An additional business-oriented domain model was already identified as useful to complement the demonstration case shown in this paper, as covers different domain concepts and patterns to apply, and a health-oriented domain model for the omics field, due to the complexity of the underlying data. Furthermore, with the application of the set of patterns proposed in this work in multiple situations, we open the door to enriching the knowledge about its suitability and also the opportunity of refining the transformations to automate them as much as possible. The systematization and automation of the transformation process increase the value of conceptual models as platform-independent models that can be transformed into different platform-specific models according to the analytical requirements. The proposed systematization helps data engineers in the complex process of model transformation, increasing the maintainability and evolution of Information Systems.

Acknowledgement

This work has been supported by FCT – *Fundação para a Ciência e Tecnologia* within the R&D Units Project Scope: UIDB/00319/2020, and by the Spanish Ministry of Universities and the Universitat Politècnica de València under the Margarita Salas Next Generation EU grant, and the MICIN/AEI/ 10.13039/501100011033 project. Co-financed with ERDF.

References

Abdelhedi, F., Brahim, A. A., Atigui, F., & Zurfluh, G. (2017). Umltonosql: Automatic transformation of conceptual schema to nosql databases. In *2017 ieee/acs 14th international conference on computer systems and applications (aiccsa)* (p. 272-279). doi: 10.1109/AICCSA.2017.76

Akid, H., Frey, G., Ayed, M. B., & Lachiche, N. (2022). Performance of NoSQL graph implementations of star vs. snowflake schemas. *IEEE Access*, *10*, 48603–48614. doi: 10.1109/ACCESS.2022.3171256

Albdaiwi, B., Noack, R., & Thalheim, B. (2014). Pattern-based conceptual data modelling. In B. Thalheim, H. Jaakkola, Y. Kiyok, & N. Yoshida (Eds.), *Information modelling and knowledge bases XXVI* (p. 21).

Almasri, N., Korel, B., & Tahat, L. (2017). Toward automatically quantifying the impact of a change in systems. *Software Quality Journal*, *25*(10), 3833-3861. doi: 10.1109/TSE.2021.3106589

Almasri, N., Tahat, L., & Korel, B. (2022). Verification approach for refactoring transformation rules of state-based models. *IEEE Transactions on Software Engineering*, *48*(3), 601-640. doi: 10.1007/s11219-016-9316-8

Blaha, M. (2010). *Patterns of data modeling*. CRC Press, Inc.

Burzynski, P., & Karagiannis, D. (2020). bee-up – a teaching tool for fundamental conceptual modelling. In *Joint proceedings of modellierung 2020 short, workshop and tools & demo papers.*

Castelltort, A., & Laurent, A. (2014). NoSQL graph-based OLAP analysis. In *Proceedings of the international conference on knowledge discovery and information retrieval* (pp. 217–224). SCITEPRESS - Science and and Technology Publications. doi: 10.5220/0005072902170224

Daniel, G., Sunyé, G., & Cabot, J. (2016). UMLtoGraphDB: Mapping Conceptual Schemas to Graph Databases. In (pp. 430–444). Springer. doi: 10.1007/978-3-319-46397-1_33

De Virgilio, R., Maccioni, A., & Torlone, R. (2014). Model-Driven Design of Graph Databases. In (pp. 172–185). Springer. doi: 10.1007/978-3-319-12206-9_14

Ehrlinger, L., Huszar, G., & Wöß, W. (2019). A schema readability metric for automated data quality measurement. *DBKDA 2019*, 12.

El Alami, A., & Bahaj, M. (2018). The migration of a conceptual object model com (conceptual data model cdm, unified modeling language uml class diagram...) to the object relational database ordb. *MAGNT Research Report (ISSN. 1444-8939)*, *2*(4), 318–32.

Galvão, J., Leon, A., Costa, C., Santos, M. Y., & Pastor, O. (2020). Towards designing conceptual data models for big data warehouses: The genomics case. In M. Themistocleous, M. Papadaki, & M. M. Kamal (Eds.), *Information systems* (pp. 3–19). Springer International Publishing.

Glaser, P.-L., Ali, S. J., Sallinger, E., & Bork, D. (2022). Model-based construction of enterprise architecture knowledge graphs. In J. P. A. Almeida, D. Karastoyanova, G. Guizzardi, M. Montali, F. M. Maggi, & C. M. Fonseca (Eds.), *Enterprise design, operations, and computing* (Vol. 13585, pp. 57–73). Springer International Publishing. doi: 10.1007/978-3-031-17604-3_4

Groger, C., Schwarz, H., & Mitschang, B. (2014). The deep data warehouse: Link-based integration and enrichment of warehouse data and unstructured content. In *2014 IEEE 18th international enterprise distributed object computing conference* (pp. 210–217). IEEE. doi: 10.1109/EDOC.2014.36

Gómez, L., Kuijpers, B., & Vaisman, A. (2020). Online analytical processsing on graph data. *Intelligent Data Analysis*, *24*(3), 515–541. doi: 10.3233/IDA-194576

Hevner, March, Park, & Ram. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75. doi: 10.2307/25148625

Huang, L., Duan, Y., Sun, X., Lin, Z., & Zhu, C. (2016). Enhancing uml class diagram abstraction with knowledge graph. In H. Yin et al. (Eds.), *Intelligent data engineering and automated learning – ideal 2016* (pp. 606–616). Springer International Publishing.

Jacobson, L., & Booch, J. R. G. (2021). *The unified modeling language reference manual.*

Karagiannis, D., & Buchmann, R. A. (2018). A proposal for deploying hybrid knowledge bases: the ADOxx-to-GraphDB interoperability case. In *Proceedings of the 51st hawaii international conference on system sciences.*

Pastor, O., España, S., Panach, J. I., & Aquino, N. (2008). Model-driven development. *Informatik-Spektrum*, *31*(5), 394–407.

Pastor, O., & Molina, J. C. (2007). *Model-driven architecture in practice: a software production environment based on conceptual modeling* (Vol. 1). Springer.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45-77. doi: 10.2753/MIS0742-1222240302

Rahayu, J., Chang, E., Dillon, T., & Taniar, D. (2000, may). A methodology for transforming inheritance relationships in an object-oriented conceptual model to relational tables. *Information and Software Technology*, *42*(8), 571–592. doi: 10.1016/S0950-5849(00)00103-8

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph databases: new opportunities for connected data.*

" O'Reilly Media, Inc.".

Santos, M. Y., & Costa, C. (2020). *Big data: Concepts, warehousing and analytics*. River Publishers.

Sellami, A., Nabli, A., & Gargouri, F. (2020). Transformation of data warehouse schema to NoSQL graph data base. In A. Abraham, A. K. Cherukuri, P. Melin, & N. Gandhi (Eds.), *Intelligent systems design and applications* (Vol. 941, pp. 410–420). Springer International Publishing.

Smajevic, M., Hacks, S., & Bork, D. (2021). Using knowledge graphs to detect enterprise architecture smells. In E. Serral, J. Stirna, J. Ralyté, & J. Grabis (Eds.), *The practice of enterprise modeling* (Vol. 432, pp. 48–63). Springer International Publishing. doi: 10.1007/978-3-030-91279-6_4

Sparks, G. (2001). Database modelling in uml. *Methods & Tools*, *9*(1), 10–23.

Ziemann, P., Hölscher, K., & Gogolla, M. (2005). From UML models to graph transformation systems. *Electronic Notes in Theoretical Computer Science*, *127*(4), 17–33. doi: 10.1016/j.entcs.2004.10.025