MBSqlE: Enabling SQL as Powerful Query Language for SysML v2 in Aviation

Alexander Ahlbrecht
German Aerospace Center (DLR)
Institute of Flight Systems
Braunschweig, Germany
alexander.ahlbrecht@dlr.de

Wanja Zaeske
German Aerospace Center (DLR)
Institute of Flight Systems
Braunschweig, Germany
wanja.zaeske@dlr.de

Umut Durak
German Aerospace Center (DLR)
Institute of Flight Systems
Braunschweig, Germany
umut.durak@dlr.de

***Abstract*—Model-Based Systems Engineering (MBSE) has emerged as a promising paradigm to manage increasing engineering complexity of safety-critical systems. One significant challenge impeding the overall benefits of MBSE is the limited data accessibility of MBSE tools. This disconnect hinders both efficient collaboration and automation. In recognition of this challenge, the second version of the Systems Modeling Language (SysML) introduces a standardized Application Programming Interface (API). However, the API's query model has some limitations. For instance, no direct graph traversal of the model is featured, leading to performance bottlenecks. To address this limitation, a data access based on the Structured Query Language (SQL) is proposed in this paper. The expressive yet mature syntax of SQL unlocks vast data extraction and model analysis possibilities. In addition, rigorously tested databases such as SQLite can serve as an intermediate model representation with potential for tool qualification in safety-critical domains.**

***Index Terms*—MBSE, SysML v2, API, SQL, Aviation**

I. Introduction

Safety-critical systems are constantly evolving. To manage the growing engineering complexity, Model-Based Systems Engineering (MBSE) has emerged as a promising paradigm. MBSE utilizes models as a primary means of design, communication, analysis, and documentation. One of the goals of MBSE is to support the interdisciplinary development by fostering information exchange between engineering disciplines [1]. In the context of avionics systems, models are used throughout the life cycle process for activities including design [2], [3], safety analysis [4], security assessment [5], optimization [6], simulation [7], configuration [8]–[10], verification and validation [11], deployment [12], and certification [13]. For all these activities, it is necessary to extract related information from the model in a machine-readable format. Even with the vast capabilities of MBSE languages such as the Systems Modeling Language (SysML), practical limitations of data interchangeability and accessibility exist [14]. For instance, exchanging information between modeling tools and engineering disciplines is often difficult or only enabled with a vendor lock-in. This is one of the challenges restricting MBSE's benefits and hindering industry adoption.

Recognizing this challenge, the second version of SysML (SysML v2) [15] introduces a standardized Application Programming Interface (API) [16]. Goals of the standardized API are to improve tool exchange and foster data accessibility and related automation. Therefore, the API defines services for accessing projects, model elements, versioning, and queries. Although the API is a major step in the right direction, it does not yet solve all problems effectively. For instance, the API's query model does not feature a mechanism for graph traversal. However, a SysML model is a highly complex graph of elements connected via relationships. For an effective application on complex system models, a powerful query service with graph traversal is essential.

To address this challenge, it is proposed to enable data management based on the Structured Query Language (SQL) which is referred to as MBSqlE in the following. SQL was selected due to its mature tooling and expressive syntax, featuring model traversals of any necessary depth via Common Table Expression (CTE) [17]. The MBSqlE concept works by utilizing the SysML v2 API services to fetch and import the model into a relational database using SQLite. SQLite is a Relational Database Management System (RDBMS) and was selected due to its embedded, platform independent, and rigorously tested nature [18]. These properties make SQLite very interesting for safety-critical domains and might even allow tool qualification. To correctly store the SysML v2 model in a relational database, a lean database schema is proposed that is able to represent all aspects of the SysML v2 model. When the SysML v2 model is included in the database, vast data extraction and model analysis possibilities are unlocked. Due to the maturity of SQL, it is possible to execute the queries from a programming language of choice and to improve data accessibility. In summary, this paper contributes by:

- Analyzing and discussing SysML v2's API capabilities,
- Enabling an SQL-based query extension (MBSqlE),
- Outlining a relational database schema for SysML v2,
- Demonstrating domain-specific applications, and
- Discussing MBSqlE's opportunities and limitations

To showcase the contributions, the paper is structured in the following manner. Initially, background information and related work about SQL, SQLite, and SysML v2 are provided in Section II and Section III. Then, the MBSqlE concept is presented in Section IV, outlining how SQL can be enabled for SysML v2 models. Afterwards, the concept is demonstrated in Section V. Complementary, Frequently Asked Questions (FAQ) are discussed in Section VI and a conclusion is provided in Section VII.

II. Background

A. *SQL and SQLite Introduction*

SQL is a well established query language used to interact with relational data with over 50 years of application in research and industry [19]. It covers various features including operators to select (SELECT FROM), filter (WHERE), combine (JOIN), and order (ORDER BY) relational data. In context of this paper, a key feature are CTE allowing powerful recursive queries and complex graph traversal [17]. Even though SQL is widely used and expressive, there are some drawbacks [20]. For instance, it can be argued that it is a difficult language to read, write, and edit [21]. For SQL, many RDBMS such as Oracle, PostgreSQL, MySQL, and SQLite can be applied[1].

In this paper, we focus on SQLite as the RDBMS. SQLite is lightweight, embedded, and platform-independent [18]. Moreover, SQLite was developed with aviation-grade testing according to DO-178B, including 100% Modified Condition Decision Coverage (MCDC) coverage [22]. Considering a safety-critical context like aviation, SQLite has promising characteristics. Due to its lightweight nature, it is one of the most widely applied RDBMS and is applied in all types of embedded devices.

B. *SysML v2 Introduction*

SysML v2 [15] is a next-generation modeling language designed to overcome many challenges of its predecessor [23]. Improvements target the precision, expressiveness, and usability of the language [24]. With regard to precision, a more formal foundation was achieved by building on the Kernel Modeling Language (KerML) [25], in turn enabling features such as the textual notation. In terms of expressiveness, additional features were added, such as views, viewpoints, variants, trade-offs, and analysis cases. Moreover, usability was improved, e.g., by introducing a consistent pattern for language constructs. Each SysML v2 element can now be expressed with definition (type) and usage (instance) [26].

Another important SysML v2 extension in the context of this paper is the addition of a standardized API [16]. A goal of the API is to improve interoperability with other tools and models [27]. Features of the API are a commit-based version control for projects and elements as well as standardized services. One service is the query service, that is defined by the query model depicted in Fig. 1.

The query model defines that it is possible to use constrained-based queries for a project, where a complex constraint can be composite or primitive constraints [16]. A primitive constraint can filter for properties and values by using defined operators. Additionally, multiple primitive constraints can be combined with a composite constraint using the join operator to perform more comprehensive searches. With the current query model, it is possible to retrieve one or a set of elements that were filtered for one or multiple properties. However, the query model does not include a feature for more complex searches that require to traverse relationships of elements.

[1]https://db-engines.com/en/ranking

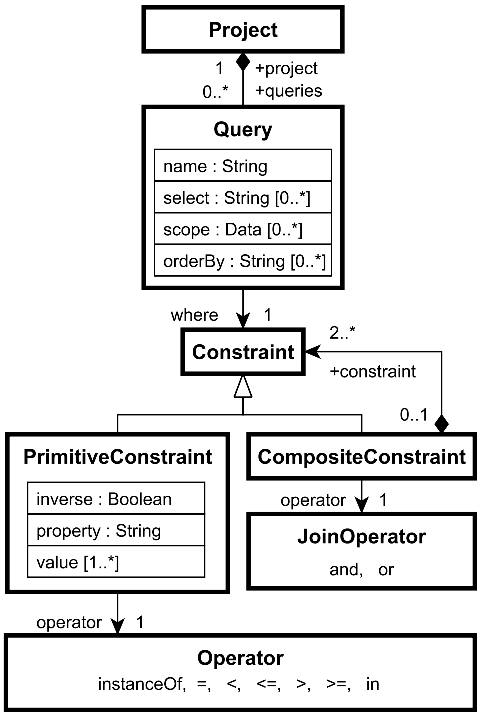


Fig. 1: SysML v2 Query Model (adapted from [16])

C. *Comparison of SysML v2's query API and SQL*

The data in SysML v2 can be described as a highly connected graph, for the standard library, we identified 200.000 elements and 16.000.000 relations between them. That means that there are roughly 80 times more relations than elements. With the majority of datapoints in the graph being relations, these connections are naturally of prime interest for many queries.

SysML v2's API allows to filter for elements which satisfy a given predicate. However, it is not possible to write a query that traverses relations. Instead, one has to implement an algorithm in an external programming language which iteratively queries the SysML v2 API to traverse the graph. Each traversal requires a new API request as shown in Fig. 2. As API requests are implemented on top of HTTP/REST, they are comparatively slow (more than 10ms per request) and the cost is multiplied by the number of hops through the graph required.

In contrast, SQL features CTE. With it, queries can be recursively defined upon a result of itself, which unlocks graph traversal directly in the database. Hence, any breadth first/depth first search through the entire graph can be expressed in one SQL query as shown in Fig. 2. Moreover, the entire query is evaluated in the SQL database software, where data access is computationally cheap and has a low latency. Thus, numerous slow HTTP/REST requests are avoided.

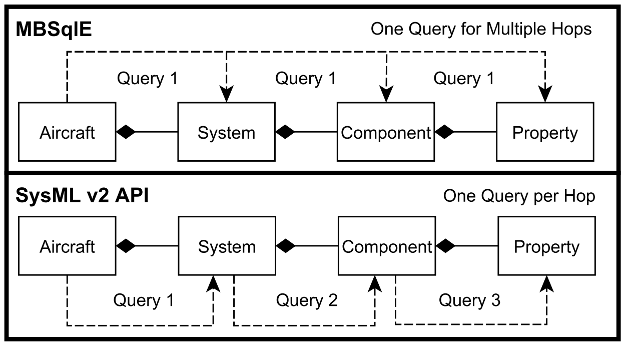


Fig. 2: Query Comparison of SysML v2 API and SQL

III. Related Work

In the context of avionics, different modeling approaches exist in the related work. Some are based on general-purpose modeling languages such as SysML [4], [11] or the Architecture Analysis & Design Language (AADL) [28]. Others focus on vendor-specific solutions such as MathWorks® tools [2], [8], [29]. Finally, some even developed domain-specific solutions [3], [13]. All of them have in common that they require a systematic data access. Although some standardized exchange formats (e.g. XML Metadata Exchange (XMI)) and query languages (e.g. Object Constraint Language (OCL)) exist, they often needed extensions for practical use. As a result, tool- or vendor-specific APIs were implemented to improve data accessibility. Vendor-specific APIs have the disadvantage that they are not standardized, can be susceptible to version changes, and sometimes lack qualitative IDEs or debugging.

For domain-specific models, domain-specific query concepts exist. An interesting avionics API example is the Essential Object Query (EOQ) language [30], which was developed to interact with the Open Avionics Architecture Model (OAAM) [3]. EOQ provides similar query capabilities as OCL, but also supports model modifications, error handling, and change events [10]. Similarly, a certification framework for aviation software was developed called CertGATE. To interact with the assurance case model, an Assurance Case Query Language (ACQL) was developed based on OCL. Another domain-specific example is the certification-focused data curation model named Rapid Assurance Curation Kit (RACK) [13]. RACK builds on a triplestore data model and supports a graphical query construction using SPARQL. Queries in RACK can be used to build and inspect assurance cases.

Although domain-specific modeling tools can come with their own query languages, they can still benefit from a systematic connection to a general-purpose language like SysML. In [31], the SysML v2 API capabilities were analyzed for avionics applications. Even if the API can be applied to retrieve, change, and upload data, performance is still a concern. An idea is to combine SysML v2 with database concepts to enable new aspects such as CI/CD [32].

Since the goal of this paper is to convert the SysML models into database formats, similar concepts in the related work are presented below. For example, [33] showcased the transformation of SysML v1 models into knowledge graphs in Neo4j to support model reuse. However, one of the challenges in [33] was that the interface to acquire model data was based on specific tool interfaces. This issue is tackled with the standardized SysML v2 API and used in concepts such as OpenMBEE[2]. OpenMBEE is an open-source and multi-layered software stack to access, store, and version model data as Resource Description Framework (RDF) format. Even if OpenMBEE presents a holistic approach with many capabilities, its multi-layered software stack might require a lot of effort to develop and maintain which might not be necessary for every application.

IV. MBSqlE Concept

A. Overview of Concept

To establish a powerful and lightweight connection from SysML v2 tools to domain-specific activities, the MBSqlE concept is proposed in this paper. Its goal is to enable a systematic SysML v2 model data storage and SQL-based data access as highlighted in Fig. 3. With this approach, models from multiple SysML v2 tools can be imported, managed, and exported. For the implementation, the target was a simple, robust, and maintainable solution that requires a minimal software stack. The SysML v2 models are fetched via the standardized API and transferred and imported into the SQLite database. After importing the model, SQL can be used from a programming language of choice to export to structured data formats or directly into domain tools. For interested readers, the details of each MBSqlE component will be elaborated in more detail in the following.

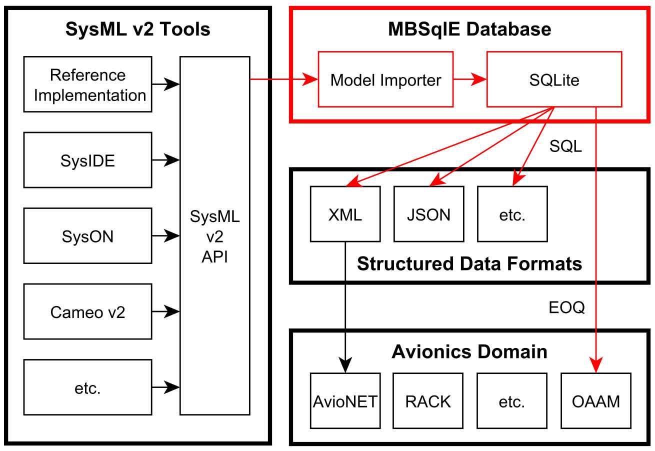


Fig. 3: Overview of MBSqlE database concept

B. MBSqlE Database

*1) **SQLite and SysMLv2 Schema**:* In order to host a SysML v2 model as data within SQLite, a database schema

[2]https://www.openmbee.org/

has to be defined. We use a skeleton schema, which is automatically filled with all possible properties defined in the SysML v2 API specification. The skeleton itself consists of three tables, *elements*, *relations* and *extended_properties* and conforms to the third normal form of RDMBS (eliminating transitive dependency). Fig. 4 depicts the skeleton schema, together with a few example property columns.

Each relation (relation ! = relationship element) in-between two elements is stored in the *relations* table. A row in said table forms a three-tuple (`origin_id`, `name`, `target_id`), establishing that there is a relation of kind `name` pointing from the origin element to the target element.

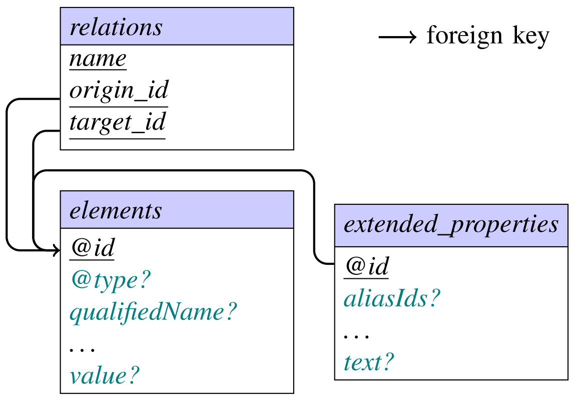


Fig. 4: Database schema with all columns of the skeleton schema and a few additional example columns. The `@` has no special semantics and is verbatim copied from the SysML v2 API's JSON-Schema, the `?` denotes a nullable column.

The *elements* table then holds each element in the model. It is important to note that relationship elements are also represented in the elements table and connected to origin and target elements via the relations table. For each element, all remaining singular element properties[3] are stored as properties in the elements table. These properties include the element's `@type`, `qualifiedName`, `value` and about 40 additional properties. If a specific element does not feature a property, then the property column for that element's row is set to `NULL`. A foreign key constraint on the *relations* table ensures that each relation's origin & target element exist within the *elements* table.

Now, all non-singular, non-relational properties are stored in the *extended_properties* table. Multiple rows in the *extended_properties* table may relate to a single element. This is what enables to store properties of composite types like arrays of texts without violating the first normal form of RDBMS (eliminating composite or multi-valued attributes). This is for example used by the `aliasIds` property, which declares an array of aliases for a given element.

[3] *remaining* as in not already covered by the *relations* table; *singular* as in the property is not a composed but a primitive type

This skeleton schema can represent the structure of elements and their associated properties found in a SysML v2 model efficiently. To fill this skeleton, we parse the JSON-Schema provided for SysML v2. Within the JSON-Schema, all possible element types and their inherent properties are enumerated. By overlaying all possible element properties and categorizing them as relational and/or singular, we get a list of columns to be inserted into the *elements* table (all non-relational, singular properties) and into the *extended_properties* table. Finally a check is inserted for the `name` column of the *relations* table, to assure that only allowed relation kinds can be inserted.

Using a skeleton schema lowers the effort for updates of the SysML v2 specification. As long as the basic structure (elements with properties) remains, any addition or removal of properties requires no changes in our code except for the automated regeneration of the SQL statements for the database initialization. The use of the element's Universally Unique Identifier (UUID)s as primary key for the elements table enables efficient updates on later imports. If the update contains an element with a UUID already present in the database, that element and all its associated data is updated based on the new element's data. Meanwhile, storing all relations in a single table simplifies graph traversal tremendously, a CTE like `WITH RECURSIVE` on the *relations* table suffices to traverse arbitrary depths of the elements graph.

To accelerate queries, we introduce individual indices on all columns of the *relations* table and the commonly used properties in the *elements* table. These include columns like `@type`, `name` and `value`. The indices allow to find any in row in the respective table by a search for the indexed column within logarithmic time.

*2) **MBSqlE Importer**:* The import is implemented in our open-source tool, `sysml-v2-sql`[4]. A list of elements to be imported in exactly the format which is described in the SysML v2 API is iterated over twice. The first iteration inserts all elements, the second completes the associated properties and relations from these elements. The following enumeration provides a detailed description of the import algorithm:

i) Fetch all elements via SysML v2 API. Our tool can fetch from any SysML v2 API server, and optionally dump the fetched data also into a JSON file.
ii) Import the exported elements, either from Random Access Memory (RAM) (where our tool fetched the data to) or from a JSON file (which may be generated from an external fetcher):
    a) Insert each element present in the export into the *elements* table. On conflict, (an element with the same UUID exists already), overwrite its properties with the new values from the export. A temporary table is created to memorize all UUIDs of elements that where updated during this import.
    b) Using aforementioned temporary table, all extended properties belonging to elements updated by this import are deleted. Finally, all relations

[4] https://github.com/DLR-FT/sysml-v2-sql

originating from those elements are deleted. Then, the temporary table can be dropped.

c) Re-iterate all elements from the export, and insert their associated relations and extended properties.

iii) During import, all observed element properties are tracked. Irregularities, such as properties that, however, have no corresponding column in the database, are logged with a warning. Relations to non-existing elements are discovered using foreign key checks. The *relations* `name` column is checked against a whitelist of known properties (e.g. definition) that can refer to other elements.

iv) After reporting any irregularities, performance tweaks are applied. These include an optional VACUUM of the tables, and a mandatory ANALYZE. The former defragments the database to achieve smaller on-disk size, while the latter updates *sqlite_stat1*. This table informs the query planner how well each index allows filtering of data, leading to efficient and fast query plans.

C. MBSqlE Query Concept

When the SysML v2 model is imported in the SQLite database, an SQL-based data access is enabled. For corresponding queries, the schema of Fig. 4 builds the foundation. All kinds of queries can be created using two atomic operations: Filtering for specific properties of elements and filtering between elements using a relation type. Simple queries filtering for element properties can mainly interact with the *elements* table of the database as shown in Listing 1. In the example query, the `declaredName`, `@type`, and `id` are selected for all elements whose `@type` contains the string ”Literal”.

```
-- Select resulting properties
SELECT declaredName, "@type",
"@id" FROM elements

-- Define property filter for elements
WHERE "@type" LIKE '%Literal%'
```

Listing 1: Element query - property filter

In contrast to simple element filters, queries with graph traversals utilize the *relations* table as shown in Listing 2. First, starting elements that are not part of a library are selected in the query. Then, relations are joined where the `origin_id` matches the `id` of one of the starting elements. When one element is the origin of multiple relations, one row for each match is created. Due to the additional filter in the `WHERE` clause, only relations with the `name` ”definition” are selected. Complementary, elements are joined matching the `target_id` of the added relations. From the resulting table, only the `declaredName` columns of the connected elements are selected and displayed.

```
-- Select resulting properties
SELECT e1.declaredName AS 'Definition',
e2.declaredName AS 'Usage'

-- Select starting elements
FROM elements as e1

-- Add relations of starting elements
LEFT JOIN relations AS r1
ON r1.origin_id = e1."@id"

-- Add related elements
LEFT JOIN elements AS e2
ON r1.target_id = e2."@id"

-- Filter for elements and relations
WHERE r1.name = 'definition'
AND e1.isLibraryElement=0
```

Listing 2: Relation query - definitions and usages

When the two presented mechanisms (element filter and relation traversal) are combined, powerful queries can be created. One example is shown in Listing 4. In the following section, a corresponding use case will be presented outlining why complex queries might be interesting in the context of aviation/avionics. Moreover, recursive queries can be build upon the presented mechanisms, e.g., to:

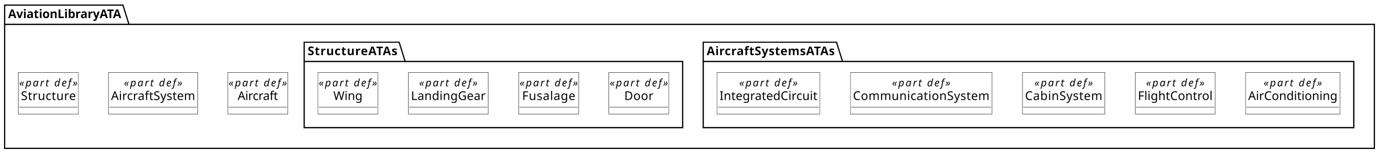
- Recursively traverse owned elements/features
- Recursively traverse element collections
- Recursively collect element parameters
- Recursively identify and use inheritance

V. Demonstrating MBSqlE

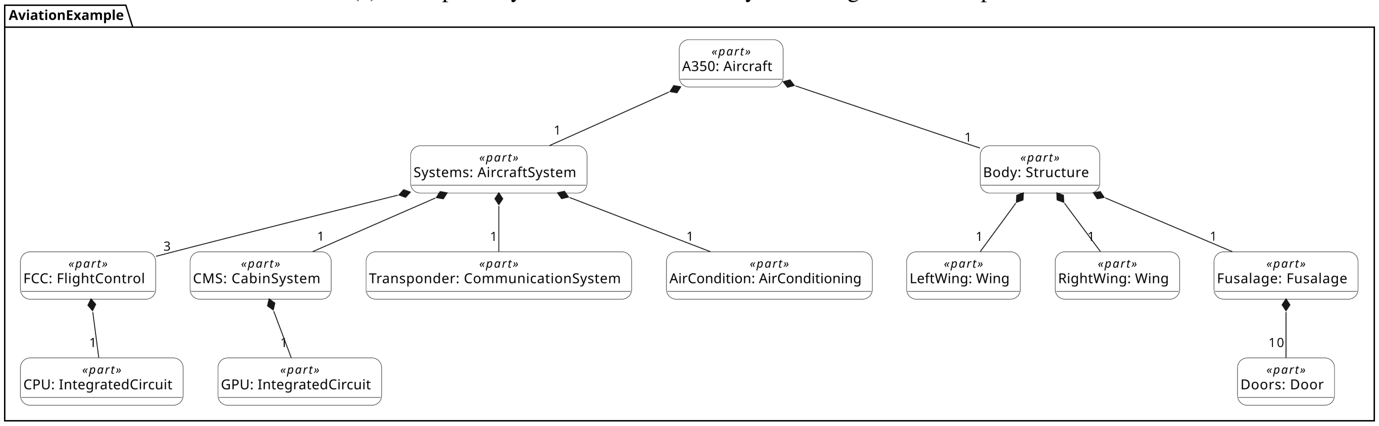
The storyline for showcasing the MBSqlE concept is the following. A highly complex and distributed aircraft development covers varies topics and involves multiple organizations. Each organization has a different focus and might only be interested in a specific aspect of the aircraft development. Still, the aircraft integration is typically coordinated by one company. To manage the complex development, a model of the aircraft is created. Now, involved organizations want to always get the newest model information for their specific area. For instance, one company might manage the structural composition while another focuses on the electric components and the related network. This is one of the use cases, where the MBSqlE concept might be interesting and will be elaborated in the following using the SysML v2 setup in Fig. 5. The example is simplified to fit into the paper, but the related concept is scalable.

A. SysML v2 Aircraft Library and Model

The first part in Fig. 5a shows a SysML v2 library defining aviation related element types. The library is organized according to the traditional ATA 100 specification [34] and covers structure and aircraft system related aspects. Generally, utilizing domain-specific libraries can assist to identify the topics that need to be covered while simultaneously improving the semantic formality of the resulting model. The second part in Fig. 5b shows a hierarchically structured model of an A350 aircraft that uses the defined library types. The structural components encompass the aircraft wing compartments, the fuselage, as well the integrated doors. In terms of aircraft systems, a Flight Control Computer (FCC), Cabin Management System (CMS), transponder, and air condition are represented in the model.



(a) Excerpt of SysML v2 aviation library according to ATA chapters


AviationExample
«part» A350: Aircraft
«part» Systems: AircraftSystem
«part» Body: Structure
«part» FCC: FlightControl
«part» CMS: CabinSystem
«part» Transponder: CommunicationSystem
«part» AirCondition: AirConditioning
«part» LeftWing: Wing
«part» RightWing: Wing
«part» Fusalage: Fusalage
«part» CPU: IntegratedCircuit
«part» GPU: IntegratedCircuit
«part» Doors: Door
1
1
3
1
1
1
1
1
1
1
1
10

(b) Excerpt of aircraft model in SysML v2 that uses library types

Fig. 5: SysML v2 demo setup with aviation library and aircraft model

Considering our use case, some companies might be interested in the structural aspects, while other might want the data of specific aircraft systems. Hence, a powerful query service would be required that automatically fetches all of the relevant model information. To enable this query service via MBSqlE, the SysML v2 model needs to be imported into the database. As a prerequisite, the model needs to be uploaded into a SysML v2 API compliant server such as the one provided by the proof-of-concept implementation[5].

B. MBSqlE Import Demonstration

For the import, the `sysml-v2-sql` tool can be applied that is elaborated in Section IV-B2. After initializing a database with the `init-db` command, the `fetch` command fetches the SysML v2 model via the standardized API. In Listing. 3, the import of the library model of Fig. 5a is shown.

```
[*] opening database "Demo.db"
[*] fetching started
[*] fetched 29 elements spread over 1 pages in 118.5013ms
[*] applying performance tweaks
[*] inserting elements
[*] inserted 29 elements over 4.584ms, averaging
     158.117µs/element ↔ 6324 elements/s
[*] inserting relations & extended_properties
[*] inserted 4 relations over 745.7µs, averaging
     186.425µs/relation ↔ 5364 relations/s
[*] inserted 266 relations over 13.1886ms, averaging
     49.581µs/relation ↔ 20169 relations/s
[*] commiting changes to db
[w] the following attributes were not always understood:
 | {
 |     "lifeClass",
 |     "visibility",
 |     "ownedConjugator",
 |     "multiplicity",
 | }
[*] resetting performance tweaks
[*] executing "ANALYZE" in db
[*] that took 6.333ms
[*] import took 29.9534ms
```

Listing 3: Console output of database import

[5]https://github.com/Systems-Modeling/SysML-v2-Release

As visible in the import log in Listing 3, the library is represented with 29 elements in the server that are connected via 270 relations and contain 0 extended properties. The server representation (relations and elements) of the model differs from the textual notation due to many implicit aspects that can be derived from the textual notation [31]. Even though one might not intuitively expect any relations in the library model of Fig. 5a, 270 relations are created in the server and logged during the import. Relation kinds include owner, feature, definition, and many others.

As shown in Listing. 3, the import per relation and element can be quite fast with the `sysml-v2-sql` tool and is in the realm of microseconds in the example. For a model import with 2×10^5 elements and 16×10^6 relations, an import average of 113 µs per element insertion and 62 µs per relation insertion was achieved, totaling the import with pre-processing and database maintenance overhead to 1063 s using a single core of an AMD 7840U processor,

During import, element properties that were not always understood are collected and listed afterwards for validation purposes. In the library example, related properties are: ”lifeClass”, ”visibility”, ”ownedConjugator”, and ”multiplicity”. As the SysML-v2 draft evolves, minor changes in the JSON-Schema manifest. These changes are reflected whenever data is imported from one version of the draft into a DB initialized for a different version. We expect this to vanish, once the standard is ratified – any remaining issues then indicate a deviation from a tool to the standard.

To finalize the import, mechanisms such as indexing are executed, since they significantly improve the database performance. In summary, for the library of Fig. 5a, the fetching of elements via the SysML v2 API took $119\,\mathrm{ms}$ while the total import took $30\,\mathrm{ms}$. In the example, the SysML v2 API server was hosted in a Docker container on the same laptop.

C. MBSqlE Query Demonstration

After importing the elements into the database, the expressiveness of SQL can be utilized to support various queries. In relation to the distributed aircraft development use case, an avionics company might want to get the relevant information about the modeled aircraft systems while a manufacturing company might want to know about the structural components.

To support the required data access, the query shown in Listing 4 is used in this paper. The query combines the element filter and graph traversal operations discussed in Section IV-C. Initially, all definitions from a specific ATA package are selected by filtering elements for the `@type` and `qualifiedName`. Then, the instances and owner elements are added by using the corresponding `name` of the relations.

As visible in Fig. 6, queries returned the element instances of the selected ATA package, the corresponding type, and the owner in a few milliseconds. For larger models with over 2×10^5 elements and 16×10^6 relations, query performance was in a similar realm (milliseconds till seconds). During these test cases, it was observed that database optimization (e.g. indexing) plays a key role in query performance. This is why a mandatory ANALYZE was added to the database import algorithm described in Section IV-B2.

With regard to the aircraft development storyline, this demonstration showcases how powerful queries can be written and executed with MBSqlE to improve data accessibility. Importantly, the simple query shown in Listing 4 is scalable. It would work with large libraries and complex system models. The more complex the queries get and the more graph traversing is necessary, the better will be the performance improvement over using the SysML v2 API's query service.

VI. Frequently Asked Questions

During the conception and implementation of MBSqlE, some questions were repeatedly asked by different stakeholders. As a result, this section was created to address the FAQs.

FAQ 1. Doesn't the SysML v2 API solve all the interoperability problems? The SysML v2 API lays important interoperability foundations. It standardizes model versioning, model exchange formats, query requests, etc. Still, it does not mean that everything is perfectly suited for every use case. In our use cases, complex and rapid graph traversal queries are of importance. Therefore, a solution like MBSqlE could be an interesting addition. It is moreover important to mention that even the MBSqlE concept benefits from the SysML v2 API. We are using the API to fetch the model into the database and are essentially providing a specific data management tooling that is enabled and compliant with the SysML v2 interoperability concept.

FAQ 2. How does the approach support aviation/avionics related development activities? Complex models for safety-critical systems and the need for powerful queries go hand in hand. Model queries are used for topics such as optimization, simulation, configuration, V&V, and many more activities. SQL is a proven solution to systematically manage large datasets. Moreover, embedded and rigorously tested RDBMS' such as SQLite exist that might even allow for development activities related to tool qualification. Even if tool qualification is not targeted, a well-tested RDBMS as model storage with powerful queries can be valuable in safety-critical developments. Similarly, the lightweight MBSqlE concept should make a deployment and maintenance in multiple companies and research organization possible.

FAQ 3. Why do we focus on a relational and not a graph database? Graph databases are a refinement of relational databases. After all, an edge in a graph is relation between two nodes. The refinement mainly concerns with efficient index structures, built-in graph algorithms and specialized query syntax for graph-patterns. Efficient index structures can be added by hand in RDBMS such as SQLite. Large databases engines like Postgres add suitable indices automatically ad-hoc, even for individual queries. Moreover, due to highly connected nature of SysML v2 models, graph algorithms become less efficient. In a test case, we tried to find the shortest path between SysML v2 elements with a graph database. Identified paths jumped over inherited features and were only helpful with precisly defined queries. In terms of qualification, built-in graph algorithms still have to undergo V&V. Hence, we believe it is easier to focus on a lightweight RDBMS implementation, then it is to qualify a big, monolithic graph database with built-in algorithms. With regard toquery syntax, we see potential for graph query languages better suited than SQL, for example SPARQL (from the RDF). However, it might be best to provide a higher level query language specific for SysML v2 models. Considering data schema flexibility (strength of graph databases), there is no need since the SysML v2 language schema is pre-defined and can be statically represented.

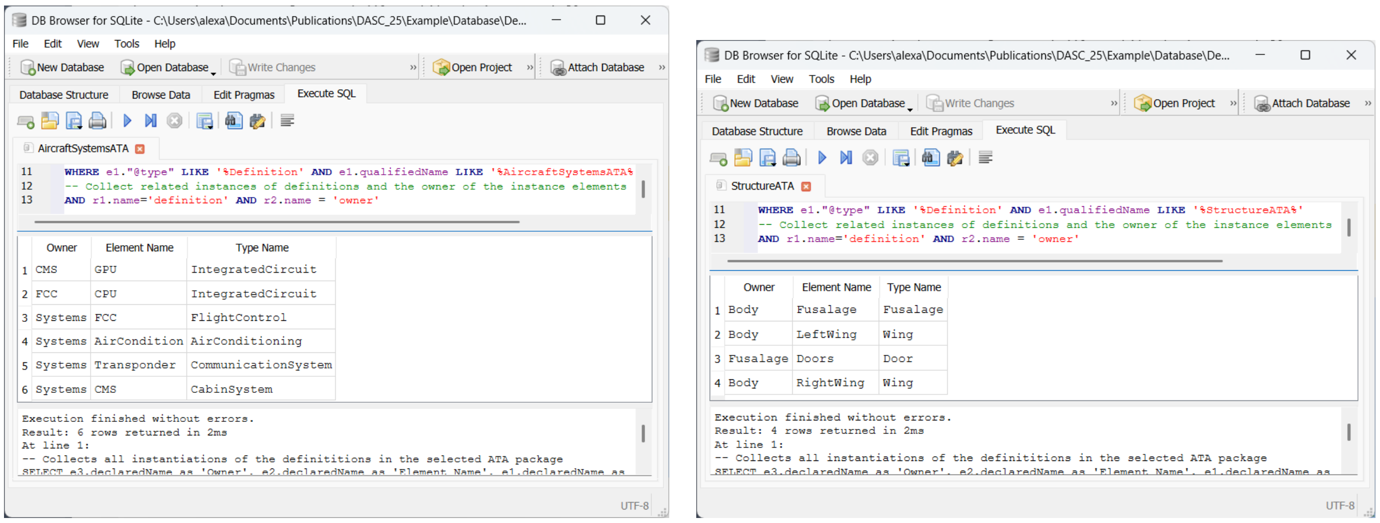
FAQ 4. How does the query performance of MBSqlE compare to SysML v2 API queries? In terms of query performance, many aspects such as query complexity, database optimization, and implementation influence the results [35], [36]. Hence, [35] highlights that it is not easy to make general statements about query performance due to many dependencies. In case of the SysML v2 API, query performance highly depends on the complexity of the query and server request latency. This is amplified due to the fact that each graph traversal requires a new query and corresponding server request. Server latency differs for local-, LAN- or VPN-setups. However, we observed queries for models with a similar complexity of the example model can take over $10\,\mathrm{s}$. This observation was one reason initiating the MBSqlE concept. Anecdotally, we can see that the slowest part of the MBSqlE concept is the elements download via the SysML v2 API with $119\,\mathrm{ms}$ as shown in Listing. 3. This download time increases

```sql
-- Collects all instantiations of the definititions in the selected ATA package
SELECT e3.declaredName as 'Owner', e2.declaredName as 'Element Name', e1.declaredName as 'Type Name'

FROM elements as e1
LEFT JOIN relations r1 ON e1."@id" = r1.target_id
LEFT JOIN elements e2 ON r1.origin_id = e2."@id"
LEFT JOIN relations as r2 ON e2."@id" = r2.origin_id
LEFT JOIN elements e3 ON e3."@id" = r2.target_id

-- Filters for definition elements in ATA package
WHERE e1."@type" LIKE '%Definition' AND e1.qualifiedName LIKE '%AircraftSystemsATA%'
-- Collect related instances of definitions and the owner of the instance elements
AND r1.name='definition' AND r2.name = 'owner'
```

Listing 4: SQL query to get instances of definitions from library package



```sql
WHERE e1."@type" LIKE '%Definition' AND e1.qualifiedName LIKE '%AircraftSystemsATA%
-- Collect related instances of definitions and the owner of the instance elements
AND r1.name='definition' AND r2.name = 'owner'
```

	Owner	Element Name	Type Name
1	CMS	GPU	IntegratedCircuit
2	FCC	CPU	IntegratedCircuit
3	Systems	FCC	FlightControl
4	Systems	AirCondition	AirConditioning
5	Systems	Transponder	CommunicationSystem
6	Systems	CMS	CabinSystem

```
Execution finished without errors.
Result: 6 rows returned in 2ms
At line 1:
-- Collects all instantiations of the definititions in the selected ATA package
SELECT e3.declaredName as 'Owner', e2.declaredName as 'Element Name', e1.declaredName as
```

```sql
WHERE e1."@type" LIKE '%Definition' AND e1.qualifiedName LIKE '%StructureATA%'
-- Collect related instances of definitions and the owner of the instance elements
AND r1.name='definition' AND r2.name = 'owner'
```

	Owner	Element Name	Type Name
1	Body	Fusalage	Fusalage
2	Body	LeftWing	Wing
3	Fusalage	Doors	Door
4	Body	RightWing	Wing

```
Execution finished without errors.
Result: 4 rows returned in 2ms
At line 1:
-- Collects all instantiations of the definititions in the selected ATA package
SELECT e3.declaredName as 'Owner', e2.declaredName as 'Element Name', e1.declaredName as
```

(a) Query result for AircraftSystemATA

(b) Query result for StructureATA

Fig. 6: ATA query applied on avionics domain model in SQLite browser

with larger models. To address this in MBSqlE, the database can only import the changed elements using the diffCommits() functionality of the SysML v2 API [16].

FAQ 5. Will the SQLite database ensure the correctness of SysML models? Basic correctness like foreign key constraints can be enforced. Triggers and check constraints can provide additional assurance, ranging from SysML soundness down to specific domain usage rules. However, triggers and checks are not yet implemented in the current MBSqlE solution.

FAQ 6. How can SQLite be qualified and what needs to be qualified? Design documents describing the nominal behavior of SQLite in detail have already been written. From these, multiple comprehensive test-suites have been developed, achieving 100% MCDC [22]. The tool qualification according to DO-330 [37] therefore mostly boils down to extracting the requirements from the design, establishing the trace from source code to requirements, and the coverage links from tests to source code. Depending on the required Tool Qualification Level (TQL), the qualification effort will vary.

VII. Conclusion

In summary, we presented a SysML v2 data management solution called MBSqlE based on the RDBMS SQLite. The goal of MBSqlE is to enable powerful SQL queries for SysML v2 models using a simple, robust, and maintainable software stack with a potential for tool qualification. Utilizing the SysML v2 API, the model is fetched and imported into SQLite. Then, powerful SQL queries can be performed to get access to model information for tasks such as optimization, configuration, V&V, or simulation. Therefore, queries can be executed from a programming language of choice and used to directly feed domain tools or output structured data formats.

A current limitation of the MBSqlE concept is that queries for SysML v2 models are not easy to write and read due to the complex nature of the SysML v2 relationships. This issue is still present when using SQL queries. Hence, future work could propose a lightweight high-level language on top of SQL for dedicated queries and easier domain-specific applications. Another topic for future work would be the combination of the MBSqlE data management concept with domain-specific SysML v2 libraries. For instance, an avionics library could be defined in SysML v2 and used to create and import domain-specific SysML v2 models. This would allow dedicated queries for the library types as highlighted in the demonstration.

Finally, some database improvements could be implemented. For instance, triggers and checks could ensure the correctness of SysML v2 models in the database. Moreover, there is the potential to support SysML v2 API features with MBSqlE such as getProjects() and getElements(). This would allow to connect the MBSqlE database to SysML v2 viewers for visual inspection of the database information. This showcases once more that MBSqlE is not necessarily a competitor to the SysML v2 API, but a data management tool extension that benefits from the SysML v2 interoperability concepts.

References

[1] INCOSE, *Systems Engineering*, n.d. [Online]. Available: https://www.incose.org/systems-engineering (visited on 02/25/2021).

[2] Y. Uludağ, Ö. Bayoğlu, B. Candan, and H. Yılmaz, "Model-Based IMA Platform Development and Certification Ecosystem," in *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*, 2023, pages 1–11. DOI: 10.1109/DASC58513.2023.10311115.

[3] B. Annighoefer, *An Open Source Domain-Specific Avionics System Architecture Model for the Design Phase and Self-Organizing Avionics*, 2019. DOI: https://doi.org/10.4271/2019-01-1383.

[4] A. Ahlbrecht, B. Lukic, W. Zaeske, and U. Durak, "A system-theoretic assurance framework for safty-driven systems engineering," *Software and Systems Modeling (SoSyM)*, 2024. DOI: 10.1007/s10270-024-01209-6.

[5] M. Werthwein and B. Annighoefer, "Model-based avionics cybersecurity framework for identification of risk and evaluation," in *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, 2024, pages 1–10. DOI: 10.1109/DASC62030.2024.10749689.

[6] M. Halle and F. Thielecke, "Bus Network Architecture- and Technology Optimisation for Avionic Systems," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, 2020, pages 1–8. DOI: 10.1109/DASC50938.2020.9256482.

[7] T. Dörr, F. Schade, A. Ahlbrecht, *et al.*, "A Behavior Specification and Simulation Methodology for Embedded Real-Time Software," in *2022 IEEE/ACM 26th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2022, pages 151–159. DOI: 10.1109/DS-RT55542.2022.9932069.

[8] B. Lukić, S. Friedrich, T. Schubert, and U. Durak, "Automated Configuration of ARINC 653-Compliant Avionics Architectures," in *AIAA SCITECH 2024 Forum*. DOI: 10.2514/6.2024-1856.

[9] M. Halle and F. Thielecke, "Model-based transition of IMA architecture into configuration data," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 2016, pages 1–10. DOI: 10.1109/DASC.2016.7777950.

[10] B. Annighoefer, M. Brunner, J. Schoepf, B. Luettig, M. Merckling, and P. Mueller, "Holistic IMA Platform Configuration using Web-technologies and a Domain-specific Model Query Language," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, 2020, pages 1–10. DOI: 10.1109/DASC50938.2020.9256726.

[11] D. Rothe, M. Rahm, C. Hagen, and A. Bierig, "Model Based Verification and Validation Planning for a Solar-Powered High-Altitude Platform," *INCOSE International Symposium*, volume 33, number 1, pages 150–162, 2023. DOI: https://doi.org/10.1002/iis2.13014.

[12] F. Schade, T. Dörr, A. Ahlbrecht, V. Janson, U. Durak, and J. Becker, "Automatic Deployment of Embedded Real-Time Software Systems to Hypervisor-Managed Platforms," in *2023 26th Euromicro Conference on Digital System Design (DSD)*, 2023, pages 436–443. DOI: 10.1109/DSD60849.2023.00067.

[13] A. Moitra, P. Cuddihy, K. Siu, *et al.*, "RACK: A Semantic Model and Triplestore for Curation of Assurance Case Evidence," in *International Conference on Computer Safety, Reliability, and Security*, Springer, 2023, pages 149–160.

[14] T. Weilkiens, "Model interoperability," in *Handbook of Model-Based Systems Engineering*, A. M. Madni, N. Augustine, and M. Sievers, Eds. Cham: Springer International Publishing, 2020, pages 1–18. DOI: 10.1007/978-3-030-27486-3_49-1.

[15] Object Management Group, *OMG Systems Modeling Language™ (SysML©) Version 2.0 Beta 2*, Specification, 2024.

[16] Object Management Group, *Systems Modeling Application Programming Interface (API) and Services*, Specification, 2024.

[17] L. Libkin, "Expressive power of SQL," *Theoretical Computer Science*, volume 296, number 3, pages 379–404, 2003, Database Theory. DOI: https://doi.org/10.1016/S0304-3975(02)00736-3.

[18] K. P. Gaffney, M. Prammer, L. Brasfield, D. R. Hipp, D. Kennedy, and J. M. Patel, "SQLite: past, present, and future," *Proc. VLDB Endow.*, volume 15, number 12, pages 3535–3547, Aug. 2022. DOI: 10.14778/3554821.3554842.

[19] D. D. Chamberlin and R. F. Boyce, "SEQUEL: A structured English query language," in *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, series SIGFIDET '74, Ann Arbor, Michigan: Association for Computing Machinery, 1974, pages 249–264. DOI: 10.1145/800296.811515.

[20] T. Neumann and V. Leis, "A Critique of Modern SQL and a Proposal Towards a Simple and Expressive Query Language," in *CIDR*, 2024. [Online]. Available: https://www.cidrdb.org/cidr2024/papers/p48-neumann.pdf.

[21] J. Shute, S. Bales, M. Brown, *et al.*, "SQL Has Problems. We Can Fix Them: Pipe Syntax In SQL," 2024, pages 4051–4063.

[22] M. Winslett and V. Braganholo, "Richard Hipp Speaks Out on SQLite," *SIGMOD Rec.*, volume 48, number 2, pages 39–46, Dec. 2019. DOI: 10 . 1145 / 3377330 . 3377338.

[23] T. Weilkiens and C. Muggeo, *THE ABSOLUTE BEGINNER'S GUIDE to SysML v2*. INCOSE UK, 2023.

[24] S. Friedenthal, "REQUIREMENTS FOR THE NEXT GENERATION SYSTEMS MODELING LANGUAGE (SysML©v2)," *INSIGHT*, volume 21, number 1, pages 21–25, 2018. DOI: https://doi.org/10.1002/inst.12186.

[25] Object Management Group (OMG), *Kernel Modeling Language* ^{TM} *(KerML*^{TM}*)*, Standard, 2024. [Online]. Available: https://www.omg.org/spec/KerML/1.0/Beta2/PDF.

[26] M. Bajaj, S. Friedenthal, and E. Seidewitz, "Systems Modeling Language (SysML v2) Support for Digital Engineering," *INSIGHT*, volume 25, number 1, pages 19–24, 2022. DOI: https://doi.org/10.1002/inst.12367.

[27] Object Management Group, *Systems Modeling Language (SysML©) v2 API and Services Request For Proposal (RFP)*, 2018.

[28] J. Delange, L. Pautet, A. Plantec, M. Kerboeuf, F. Singhoff, and F. Kordon, "Validate, simulate, and implement ARINC653 systems using the AADL," in *Proceedings of the ACM SIGAda Annual International Conference on Ada and Related Technologies*, series SIGAda '09, Saint Petersburg, Florida, USA: Association for Computing Machinery, 2009, pages 31–44. DOI: 10.1145/1647420.1647435.

[29] M. Halle and F. Thielecke, "Avionics Next-Gen Engineering Tools (AvioNET): Experiences With Highly Automised and Digital Processes for Avionics Platform Development," in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*, 2021, pages 1–8. DOI: 10.1109/DASC52595.2021.9594509.

[30] B. Annighoefer, M. Brunner, B. Luettig, and J. Schoepf, "EOQ: An Open Source Interface for a More DAMMMMN Domain-specific Model Utilization," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2021, pages 483–492. DOI: 10.1109/MODELS-C53483.2021.00075.

[31] A. Ahlbrecht, B. Lukic, W. Zaeske, and U. Durak, "Exploring SysML v2 for Model-Based Engineering of Safety-Critical Avioncis Systems," in *2024 IEEE/AIAA 43rd Digital Avionics Systems Conference (DASC)*, 2022, pages 1–8.

[32] M. J. Schwaiger, "Versioning of SysML v2 for simulation by using Graph Databases in a CI/CD process," in *2023 International Conference on Modeling, Simulation & Intelligent Computing (MoSICom)*, 2023, pages 65–70. DOI: 10.1109/MoSICom59118.2023.10458843.

[33] C. Fu, J. Liu, and S. Wang, "Building sysml model graph to support the system model reuse," *IEEE Access*, volume 9, pages 132 374–132 389, 2021. DOI: 10.1109/ACCESS.2021.3115165.

[34] Airlines for America, *Spec 100: Manufacturers' Technical Data, Revision 1999*, 1999. [Online]. Available: https://publications.airlines.org/products/spec-100-manufacturers-technical-data-revision-1999.

[35] P. Kotiranta, M. Junkkari, and J. Nummenmaa, "Performance of Graph and Relational Databases in Complex Queries," *Applied Sciences*, volume 12, number 13, 2022. DOI: 10.3390/app12136490.

[36] F. Holzschuher and R. Peinl, "Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j," in *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, series EDBT '13, Genoa, Italy: Association for Computing Machinery, 2013, pages 195–204. DOI: 10.1145/2457317.2457351.

[37] RTCA, *Software Tool Qualification Considerations, RTCA/DO-330*, Standard, 2011.