A Graph Database Design for Multi-Domain Model Management

EngD THESIS REPORT

A Graph Database Design for Multi-Domain Model Management

Mohammad Ibrahim

March/2023

Department of Mathematics & Computer Science

EngD SOFTWARE TECHNOLOGY

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

A Graph Database Design for Multi-Domain Model Management

Mohammad Ibrahim

March 2023

Eindhoven University of Technology
Stan Ackermans Institiute – Software Technology

EngD Report: 2023/011

Confidentiality Status:
Public

Partners

Eindhoven University of Technology

Steering Group Prof.dr. M.G.J. van den Brand
H.M. Muctadir, EngD

Date March 2023

Composition of the Thesis Evaluation Committee:

Chair:	Prof.dr. M.G.J. van den Brand
Members:	H.M. Muctadir, EngD
	Dr.ir. L.G.W.A. Cleophas
	Dr. I. Kurtev

The design that is described in this report has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct.

Contact Address
Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 5.072, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31 402743908

Partnership
This project was supported by Eindhoven University of Technology.

Published by
Eindhoven University of Technology
Stan Ackermans Instituite

EngD-report
2023/011

Preferred reference
A Graph Database Design for Multi-Domain Model Management. Eindhoven University of Technology, EngD Report 2023/011, March 2023

Abstract

To remain relevant in a highly competitive marketplace, modern high-tech systems are becoming increasingly complex, enabling advanced use cases, ease of usage, and intelligent capabilities. Implementing various advanced features requires collaboration between engineering disciplines. These cross-domain collaborations, however, are often insufficient, resulting in silos where different information is available to other people. Therefore, the high-level global view of the system is often incomplete.

Similar things happen to implement high-tech systems through **Model-Based Design (MBD)**. In this case, multidisciplinary teams work with different models and modeling tools. Ensuring consistency among these models is a complex task in model management. It becomes even more complicated in multi-tool and multidisciplinary settings. Although these model artifacts are often stored in the same repository, different teams work with other parts, and thus information becomes localized. As a result, bugs and inconsistencies can go undetected until appearing later in the product's lifecycle, which is often very expensive to solve.

In this project, we aimed to develop a model management tool that extracts relationships among multi-tool models and stores them using a graph database. Later, we queried this database for more insights, for example, identifying inconsistencies, analyzing change propagation among model elements, detecting hotspots, analyzing relationship dependencies, etc., among dependent models and their elements.

Keywords
Graph Database, Multi-Domain Model, Model Management, Neo4j, NeoDash, Graph Data Visualization, EngD, Software Technology

it assume any legal liability, whether direct or indirect, or responsibility for the accuracy, completeness, or usefulness of any information.

Trademarks

Product and company names mentioned herein may be trademarks and/or service marks of their respective owners. We use these names without any particular endorsement or with the intent to infringe the copyright of the respective owners.

Foreword

It is my great pleasure to introduce the impressive work of Mohammad Ibrahim, an engineering doctoral student who has developed a graph-based model management tool. Mohammad's tool can read dependency information, along with several other properties, from popular modeling tools such as Simulink & Rhapsody and store this data in a Neo4j graph database.

Model-Based Systems Engineering (MBSE) has become increasingly popular in recent years as a way to improve the efficiency and effectiveness of system design and development. With its ability to provide a more comprehensive view of a system, MBSE has enabled engineers to design and develop complex systems more efficiently, while also improving system quality and reducing costs. As a result, MBSE has become a more popular choice for organizations that want to stay competitive in today's fast-paced market. From aerospace to healthcare, MBSE is being used to model and design complex systems across a variety of industries, making it a critical tool for modern engineering. However, managing these models and understanding the dependencies among them can be a challenging task, particularly when dealing with multiple tools and platforms.

Mohammad's work provides a valuable solution to this challenge by developing a tool that can manage complex block-based models with ease. By allowing dependency information to be read from Simulink and Rhapsody and stored in a Neo4j graph database, this tool can help researchers and engineers to better manage and analyze complex models, improving productivity and advancing our understanding of various systems.

Mohammad's research is an excellent demonstration of his expertise and dedication to the field of engineering. His graph-based model management tool demonstrates the possibilities of a graph-based approach and how it can be used in managing consistencies among complex models, and we are excited to see how it will be utilized in the future.

On behalf of the academic community, I congratulate Mohammad on his impressive achievements and wish him continued success in his professional pursuits. His work will undoubtedly benefit future research activities.

Hossain Muhammad Muctadir, EngD
PhD Candidate
Software Engineering and Technology cluster
Eindhoven University of Technology, The Netherlands
Date: March 16, 2023

Preface

This report summarizes the "A Graph Database Design for Multi-Domain Model Management" project that Mohammad Ibrahim carried out at the Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands.

This project was fully funded and initiated by TU/e as the final graduation project of the Engineering Doctorate (EngD) in Software Technology (ST) program. It was part of the Software Engineering and Technology cluster of TU/e. EngD ST program is a two-year technical designer program provided by TU/e under the banner of 4TU.School of the Stan Ackermans Institute.

This project aimed to design and develop a model management tool using a graph database. We extracted model elements, their relationships, and dependencies and stores them using graphs. Moreover, we designed and developed a graph data model that presents models and their relationships. This tool helps design engineers to access more insights about the system they are working with by querying information from the database.

This report is organized from an explanation of the problem and requirements analysis to the problem's solution, design, and implementation. This report is intended for different readers who have other interests. Readers who want to understand the project context, stakeholders, problem domain, and problem analysis can refer to Chapters 1 to 4. The ones interested in the detailed requirements, corresponding solutions, and reason behind the solutions can go through Chapters 5 to 7. Those who want to study verification and validation can read Chapter 8. Chapter 9 is for those who are interested in the management of this project. Readers interested in the summary of the achievements in the project, suggestions for future work, and the trainee's self-reflection can read Chapter 10.

Mohammad Ibrahim
March 2023

Acknowledgments

The last nine months with this project have been simply incredible. This project would not have been possible without the support of several people. To begin with, I would like to thank my supervisor, Prof. dr. M.G.J. van den Brand, for giving me this opportunity, especially to work under his direct supervision in this project. I want to thank him for his constructive feedback, encouragement, and continuous support throughout the project. His critical thinking and ideas encouraged me to explore the task differently. Without his constant assistance, this project would not have been a success.

I want to thank Hossain Muhammad Muctadir, my project client, for his continuous support, knowledge, guidance, and expertise that I needed during this project. I could not have achieved the results and deliverables without his help. Apart from my project client, I would like to thank David A. Manrique Negrin and Ion Barosan for their assistance.

I want to express my gratitude to Yanja Dajsuren for the opportunity to be part of the EngD program and especially for providing me a chance to complete my graduation project. Otherwise, I would not be able to come to this stage. I want to thank Desiree van Oorschot and all the coaches for their encouragement and support over the last two years, which has allowed me to grow professionally and personally. Also, I want to thank Dr. Judith Strother, the EngD ST professor of Corporate Report Writing, for her excellent feedback and valuable suggestions.

Finally, I extend gratitude to my beloved wife, Nishat Subah Mohana, for her love and support. Also, I would like to show my deep appreciation to my parents, my parent-in-law, and my family members in Bangladesh for their support. I thank my colleagues, the Software Technology Trainees of 2020, for our two years of teamwork, support, fun, and friendship. Moreover, I want to thank my friends Jobaer Islam Khan, Lamisha Rawshan, and Arnab Kumar Shil.

Mohammad Ibrahim
March 2023

Executive Summary

Systems have become more complex and extensive due to technological advancements in recent years. These lead to system developers not being able to comprehend complex systems. Model-Based Design (MBD) provides a mathematical and visual approach to developing complex systems. Engineers from different teams work with other modeling tools. Therefore, collaboration among various engineering disciplines is essential for implementing a complex system through model management. Also, model management ensures consistency among these models.

The goal of this project was to develop a model management tool. Firstly, we extracted model elements, their relationships, and their dependencies. We developed an interface and necessary backend infrastructure using a graph database for querying information from a repository where various cross-domain heterogeneous models are stored. Also, we designed and developed a graph data model that represents model elements and their relationships. This tool can query data related to the implementation artifacts allowing the engineers access to more insights into the system they work with.

To address the project's goal, firstly, we investigated how to get data from models created with different modeling tools. Secondly, we identified essential elements of each model and developed a parser to extract specific information. Also, we developed a data loader to store extracted data in a graph database. Thirdly, we designed a graph database for storing model information with a graph data model (meta ontology). Fourthly, we developed a visualization tool to display data from the graph database. Finally, we verified and validated the design and implementation of the tool against the requirements. This tool will assist design engineers to identify more insights, especially model inconsistencies.

Table of Contents

List of Figures

List of Tables

1.Introduction

The system complexity has increased with recent technological advancements. In many cases, models are used to describe systems. These models become larger and more complex due to the size and complexity of the system. Therefore, model management becomes more important when organizing and maintaining models for ensuring consistency [1]. It becomes more urgent when multidisciplinary teams work on models from different domains. In this project, we propose a solution to the problem of managing models. This chapter provides an overview of the project context in Section 1.1. Finally, this chapter presents an outline of this report in Section 1.2.

1.1 Project Context

Systems are becoming more complex with the expansion and development of technology, system engineering, and customer demands for high-tech products [2]. Also, there exists a strong relationship between the various disciplines because they integrate with hardware (electronics, machinery) and software components. The cost and difficulty of developing a product can be affected by these relationships. The Model-Based Systems Engineering (MBSE) approach is progressing and leading the way and is expected to become a standard practice in systems engineering [3]. An MBSE methodology focuses on modeling as its primary artifact, and its benefits include faster development, earlier system analysis, and more manageable complexity. It is used in many domains, including Robotics, Automotives, and Software Systems.

A heterogeneous system is a system that consists of components from different domains. For example, a mechatronic system includes mechanical, electrical, and computer components [4]. Each part belongs to a particular discipline that requires specific domain knowledge to interpret its purpose and function. The complexity of developing heterogeneous systems increases when engineers from different expertise and domains combine their models [5]. A model represents reality, an abstraction of things relevant to stakeholders described clearly and unambiguously. For developing models, several tools are available, including MathWorks Simulink, IBM Rhapsody, OpenModelica, LabVIEW, Enterprise Architect, and UML [33]. This project focuses on block-based models and modeling tools to see how they are interconnected. Figure 1.1 shows an example of multi-tool modeling. Here, a Rhapsody SysML model uses another model of Simulink. The interfaces (inputs/outputs) are marked as circles by which these two multi-tool models are connected and pass information to each other.

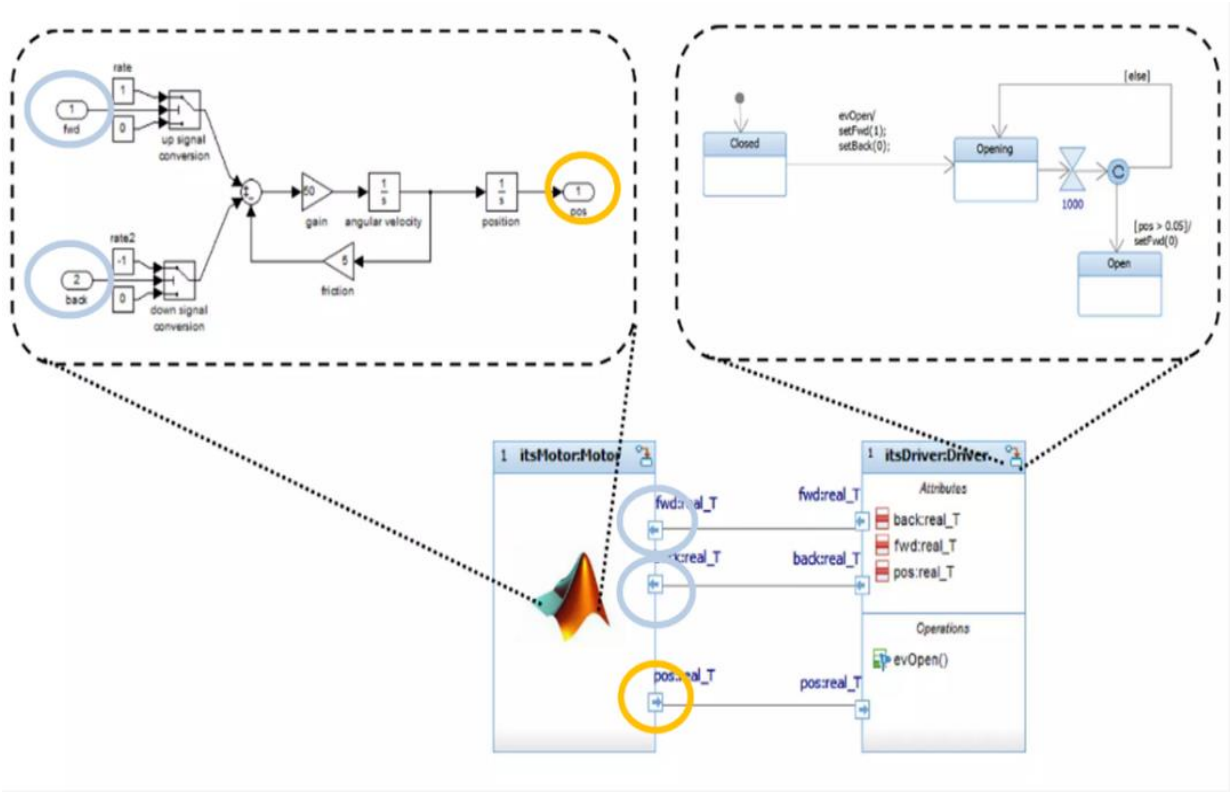

Figure 1. 1: Multi-tool modeling — Importing and Executing a Simulink Model in Rhapsody

When multidisciplinary teams work on the same system, they use different models with overlapping semantic definitions [6]. Although these artifacts are often stored in the same repository, different teams work with other parts, thus making the information localized. Consequently, bugs and inconsistencies can be undetected until later in the product's life cycle, which is often quite expensive to fix. A robot is an example of such a complex multi-domain system. Combining models created by mechanics, electronics, and software experts may be necessary. These models might be created using domain-specific tools of each domain.

Model changes in one domain can affect models from other disciplines, resulting in input/output inconsistencies throughout the system. Inconsistencies can occur at many stages throughout the entire life cycle of the development of a system. The sooner an inconsistency is detected, the cheaper it will be to fix [7]. Inconsistencies can lead to catastrophic events such as the following two examples. In 1999, NASA's unmanned MARS Climate Orbiter [8] was destroyed due to inconsistent metric units, and Airbus had a 6-billion-dollar loss due to varying specifications in different versions of design tools in 2006 [9].

Managing models belonging to the same domain may not be a complex task because of the features provided by the development tools. However, managing interrelated models from different domains can be challenging [10]. Therefore, model management recognizes and maintains models and ensures consistency. The structure and development of model management may include a model repository, modeling tools, and a database for storing model information.

In this project, we proposed a model management tool that uses a graph database to store the relationships between models to solve the problem of input/output inconsistency. We proposed to design a graph data model to specify how data is structured and stored in the graph database. This design contained the metadata of the model and the relationships between the models. This tool can be used to query artifacts for gaining more insights and identifying inconsistencies and dependencies among models earlier.

1.2 Report Outline

The rest of the report is organized as follows:

- Chapter 2 - Stakeholder Analysis: We describe the stakeholders of this project, their concerns, and communication way in this chapter.
- Chapter 3 - Domain Analysis: We provide information about the domain analysis to understand better where the problem resides.
- Chapter 4 - Problem Analysis: We refer to the problem definition and project goal and scope to realize the current situation based on domain analysis in this chapter.
- Chapter 5 - Requirement Analysis: This chapter defines the requirement elicitation and the functional and non-functional requirements.
- Chapter 6 - System Architecture and Design: We illustrate the architecture and design for developing the tool.
- Chapter 7 - Implementation: This chapter explains the tool's implementation in more detail.
- Chapter 8 - Verification and Validation: In this chapter, we describe the process of verifying and validating our implementation.
- Chapter 9 - Project Management: We present an overview of how this project was managed, including project planning, communication, and risk management.
- Chapter 10 - Conclusion: We conclude the project with results, open directions for possible future recommendations, and self-reflection.

2.Stakeholder Analysis

This chapter investigates and describes the stakeholders who can influence the project's success.

This project is an internal project of TU/e. Therefore, all the stakeholders were from TU/e. They were responsible for ensuring that the project deliverables satisfied all the requirements. They helped to maintain the quality of the project and provided their support. Table 2.1 lists the stakeholders and their concerns about this project.

Table 2. 1: Stakeholders and their concerns

Name	Role	Communication Way	Concerns
Mark van den Brand	TU/e Supervisor	▪ Biweekly update meetings ▪ PSG meetings	• Ensuring the project progress based on milestones and plan • Providing the required guidelines for project success • Ensuring the quality of the project results • Monitoring the progress of the EngD trainee • Ensuring well-written final reports
Hossain Muhammad Muctadir	Project Client	▪ Weekly update meetings ▪ PSG meetings ▪ On-demand meetings when required	• Defining the project requirements and deliverables • Helping and guiding the trainee with domain knowledge • Monitoring progress based on the defined plan • Developing an effective tool • Ensuring trainee's progress based on observations • Completing the project on time
David Manrique Negrin	Domain Expert	▪ On-demand meetings when required	• Transferring domain knowledge to the trainee • Helping with technical guideline
Ion Barosan	Domain Expert	▪ On-demand meetings when required	• Discussing and gathering domain-specific knowledge • Helping with technical guideline
Yanja Dajsuren	Program Director, EngD ST	▪ TU/e comeback days ▪ On-demand meetings when required	• Ensuring successful collaboration with project client • Ensuring that the project meets the quality requirements • Evaluating and suggesting appropriate personal and professional development of the trainee • Providing the final report's quality meets the program standards

			• Ensuring trainee's graduation
Mohammad Ibrahim	EngD ST trainee		• Providing an effective tool that meets stakeholders' expectations • Finishing the project during the specified period • Boosting technical skills by delivering the high-quality project • Enhancing the design and leadership skills • Writing the final report that meets the program standards as well as the project quality

3.Domain Analysis

At the beginning of this project, the main focus was to understand the context and identify related artifacts, also known as domain analysis. In this section, we present the domain concepts that are relevant and useful for this project.

3.1 Model-Based System Engineering

Model-Based Systems Engineering (MBSE) is an approach that applies models to support the whole system lifecycle. The International Council of Systems Engineering (INCOSE) defines MBSE as the application of modeling throughout the project life cycle, including requirements, design, analysis, verification, and validation [11]. MBSE aims to create a model of the system under development. The entire system can be defined and understood by covering it in detail [12]. Modeling provides additional benefits, such as analyzing complex technical interactions within a system. Therefore, the application of MBSE has increased in recent years, and new challenges and problems have emerged [13].

3.2 Model-Based Design

Model-based design (MBD) is a mathematical and visual method of developing complex control, communication, and signal processing systems [14]. It provides an efficient way to establish a common framework for communication throughout the design process. This methodology differs significantly from traditional design methods. A model-based design method uses continuous-time and discrete-time building blocks to define models with advanced functional characteristics rather than relying on complex structures and extensive software code. These models can facilitate rapid prototyping, software testing, and verification using simulation tools.

In the past, design engineers relied heavily on mathematical models and text-based programming. The process of developing these models, however, was time-consuming and highly error-prone. Debugging text-based programs is also a tedious process, requiring a lot of trial and error before producing a fault-free model. Conversely, model-based design tools are aimed at improving these aspects of design using model elements or blocks. In these tools, the design process is broken down into hierarchies of individual design blocks, reducing the complexity of the design process. We call these models.

We can use several tools for modeling. In this project, we were mainly interested in block-based modeling tools. We primarily identified Simulink and Rhapsody SysML tools and partially focused on OpenModelica. We discuss these modeling tools in the following sections.

3.2.1. Simulink

Simulink is an extension of MATLAB used for graphical modeling, simulation, and model-based design of multi-domain dynamic systems [15]. Simulink represents mathematical models of physical systems graphically as block diagrams. It supports system-level design, simulation, automatic code generation, and continuous testing and verification of systems. Simulink and MATLAB are tightly integrated. Therefore, users can incorporate MATLAB algorithms into their models and export simulation results into MATLAB.

Simulink has two major types of elements:

Blocks

Using blocks, we can create, modify, combine, output, and display signals. The Simulink library consists of several general types of blocks for example sources, sinks, continuous, discrete, math operations, ports & subsystems, etc.

Lines
Lines are used for transferring signals from one block to another. A line transmits signals in the direction indicated by an arrow. Lines must continuously transmit signals from one block's output terminal to another's input terminal. Figure 2.1 shows a simple Simulink model.

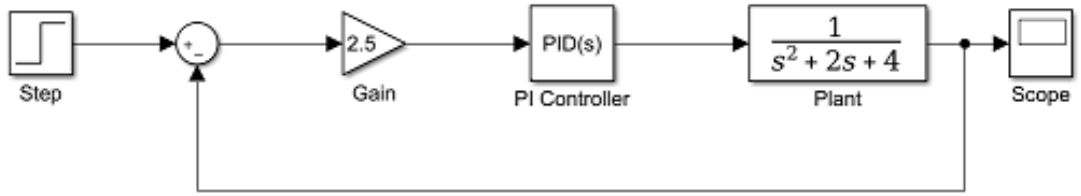

Figure 3. 1: A simple Simulink Model

3.2.2. IBM Rhapsody - SysML
IBM Rational Rhapsody is a visual development environment for building real-time or embedded systems and software. As of this point, we refer to IBM Rational Rhapsody as Rhapsody. It supports various modeling languages, including UML, SysML, AUTOSAR, MARTE, DDS, MODAF, etc. [31]. In this project, we use IBM Rhapsody as System Modeling Language (SysML) modeling tool.

SysML

The Systems Modeling Language (SysML) is a standard, general-purpose, modeling language for model-based systems engineering (MBSE) [16]. SysML supports the specification, analysis, and design of a broad range of complex systems such as control systems. It was developed by Object Management Group, Inc. (OMG). It is defined as an extension to UML based on UML's profile mechanism. SysML can specify, analyze, design, verify, and validate complex hardware and software systems using these extensions. Specifically, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametric.

There exist nine kinds of SysML Diagrams [17]:

- **Block Definition Diagram (BDD)**: BDDs display elements such as blocks and value types. In a BDD, blocks represent classes, as in a UML diagram.
- **Internal Block Diagram (IBD)**: The IBD diagram specifies the internal structure of a BDD block. Additionally, it shows the connections and interfaces between all parts in a single block.
- **Use case diagram**: In SysML, use case diagrams are like UML diagrams. these give a graphical overview of all the system's functionalities and the actors that can utilize them.
- **Activity diagram**: It is a diagram of behavior that specifies behavior through actions, inputs, and outputs.
- **Sequence diagram**: An operational call and signal diagram can show how parts of a block interact with one another.
- **State machine diagram**: A behavior diagram represents certain states of a system where a system can change conditions using events.
- **Parametric diagram**: Constraints are linked to system properties in a parametric diagram.

- **Package diagram**: A package diagram is a structural diagram that shows how a system's hierarchy is structured. It is used to describe a system at a high level.
- **Requirements diagram**: A requirements diagram is a diagram that contains all the requirements for a particular system. Each of these requirements can have relationships with one another.

A good overview of all the categories and relationships between SysML diagrams is presented in Figure 3.2.

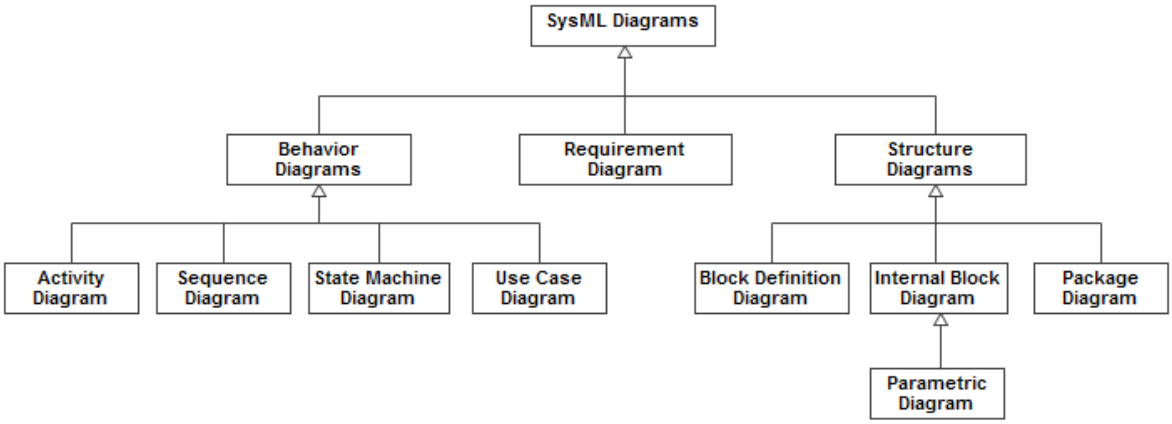

Figure 3. 2: SysML Diagram Hierarchy

Figure 3.3 shows a SysML Block Definition Diagram. It contains blocks which are the central concept in a SysML model. Blocks are connected with association, aggregation, composition, dependency, and generalization.

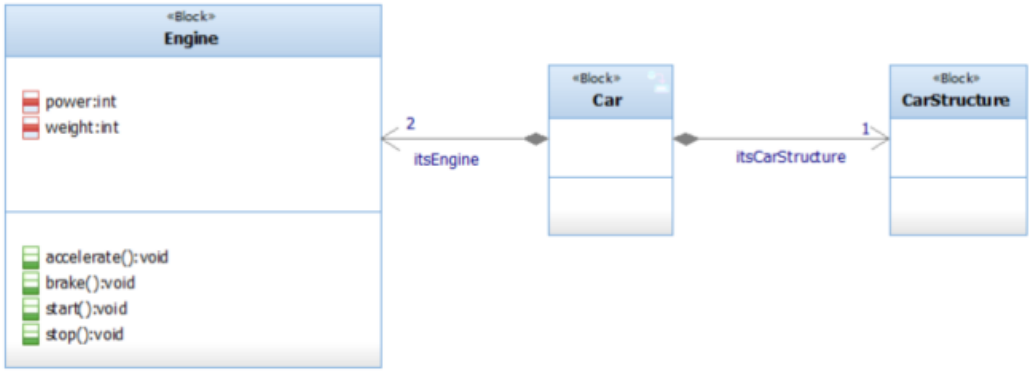

Figure 3. 3: Example Block Definition Diagram – SysML

3.2.3. OpenModelica

OpenModelica is a Modelica-based open-source framework for modeling, analyzing, and simulating dynamic systems. It was developed by the Programming Environments Laboratory (PELAB) at Linköping University, and a non-profit organization called the Open Source Modelica Consortium (OSMC) has supported it [18]. OpenModelica aims to provide a flexible and comprehensive model, compilation, simulation, and systems engineering environment for research, teaching, and industrial use.

There are several subsystems in the OpenModelica environment. OpenModelica Compiler (OMC) translates Modelica models into C code, which is compiled and executed to simulate the model.

OpenModelica Connection Editor (OMEdit) is a graphical and textual connection editor for component-based model design. It includes browsing the Modelica standard library, simulating, analyzing simulations, and presenting documentation. Figure 3.4 shows a simple feedback control model developed in OpenModelica.

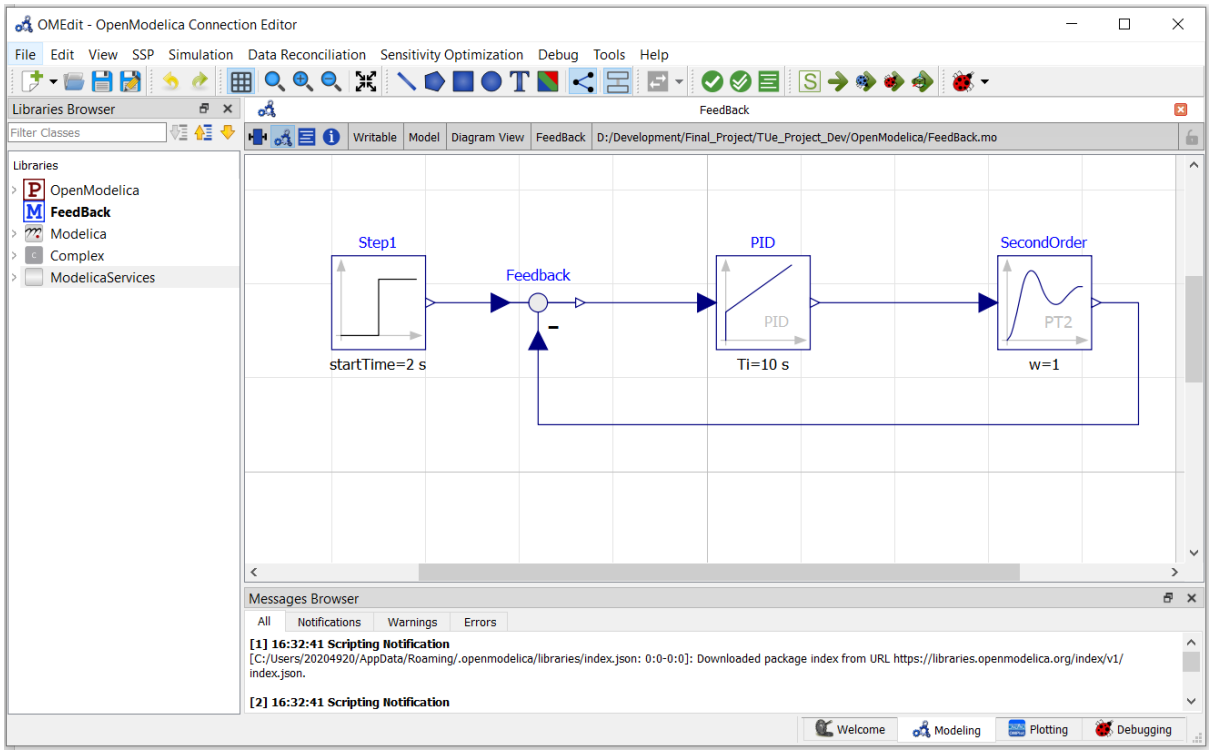

Figure 3. 4: A simple feedback control model in OpenModelica

3.3 Model Management

Modeling and simulation tools are increasingly used for industrial applications [19]. These tools support different steps in the modeling and simulation lifecycle, including defining requirements, creating models, simulating models, checking models, and writing code. However, modern industrial products' heterogeneity and complexity often require combining models from various domains. A seamless exchange of models between different modeling tools is essential for integrating a complex product model throughout the development process. Model management is a process of managing models. It ensures consistency among the models in a multi-tool and multidisciplinary setting. Graphs can represent relationships between models, similar to the structure of (meta) models.

3.4 Graph Databases

Graph databases are based on mathematical graph theory. A graph database is a collection of nodes and edges. Each node represents an entity (such as a person) and each edge represents a connection or relationship between two nodes. It stores information in the nodes (or vertices) and edges (or relationships) of a graph, as shown in Figure 3.5(b). In graph databases, relationships are stored directly with nodes, which is a valuable feature. Therefore, foreign keys and joins are not required as in relational databases. This makes it possible to read highly connected data very fast. In read access, individual data records (nodes and edges) are read in place rather than being searched globally. Thus, execution time only depends on the depth of the traversal.

According to Robinson et al. [20], graph databases are database management systems that allow users to create, read, update, and delete graphs exposed by graph data models. The graph data model is

focused on relationships, which is why graph databases are relevant to this study. Despite the name, traditional relational databases require foreign keys to infer relationships between entities. Therefore, graph databases are more suitable for problems that rely on relationships between entities. We can build models closely related to our problem of model management by using graph databases. In a graph, domain entities are represented as vertices and their relationships as edges. There are a variety of types of vertices, edges, and relationships between graphs that can be abstracted into meta-graphs. This meta-graph is known as a graph data model.

Figure 3.5(a) shows an example in which a *graph data model* contains a movie vertex, an actor vertex, and a director vertex. These are connected by ACTED_IN and DIRECTED edges, respectively. Movie, Actor, Director, ACTED_IN, and DIRECTED labels indicate vertex or edge type.

Figure 3.5(b) illustrates a graph instance of this *graph data model* and contains five actor vertices, one director vertex, and one movie vertex. The five actors' vertices are connected to the movie vertex, each with an ACTED_IN edge. The one director vertex is connected to the movie vertex with a DIRECTED edge. These vertices contain the actor and director names and the movie's title. The actor, director, and movie labels on the vertices are ignored to keep things short.

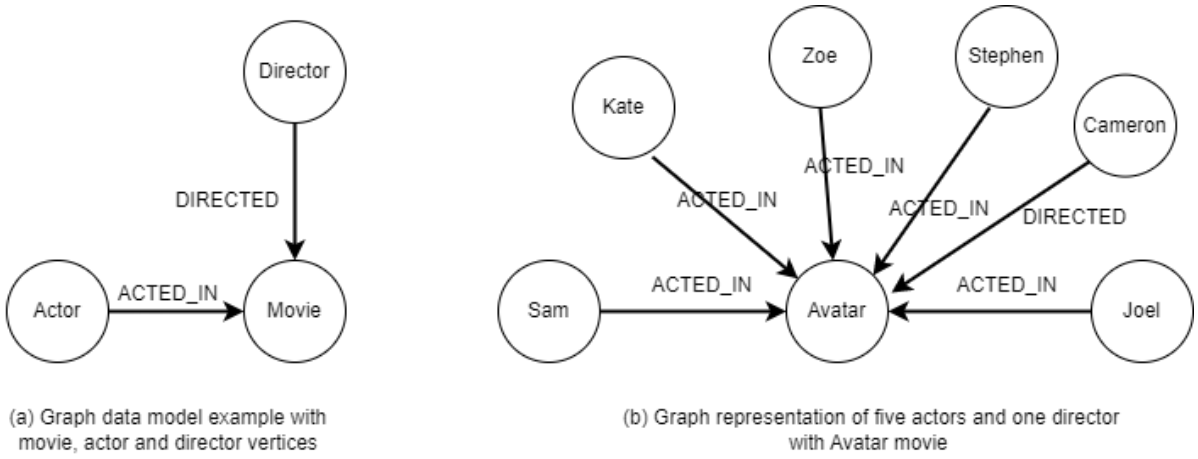

Figure 3. 5: An example of a graph and its corresponding graph data model

3.5 Neo4j

Neo4j is a graph database platform developed by Neo4j, Inc. It is a native graph database platform built to efficiently store, query, analyze, and manage highly connected data [21]. It is also an ACID (Atomicity, Consistency, Isolation, Durability) compliant transactional database. Furthermore, Neo4j is schemaless, meaning no metadata must be defined before user data is inserted. Neo4j also uses property graphs [22], indicating that vertices and edges can have properties and need to be determined by at least one label. In property graphs, relationships may have a direction that helps to identify relationship dependencies.

Moreover, Neo4j supports graph scalability, high availability, clustering, cloud graphs, a large active community, and integrated ETL. Neo4j implements the declarative graph query language Cypher, a powerful and expressive language for querying different insights from stored data. We selected Neo4j as the graph database management system for this study because it is the most widely adopted graph database [23] and comes with tools that are used to manipulate the data and visualize the stored graph.

There are many advantages of the Neo4j database. These are:

- **Performance**: The performance remains high despite significant increases in data volume.
- **Flexibility**: Data structure can be upgraded without affecting existing functionality.
- **Agility**: The data store can evolve along with the application.

4.Problem Analysis

This chapter provides an introduction to the problem this project explores. Section 4.1 briefly discusses the problem definition. Section 4.2 addresses the goals that need to be addressed and the project's scope. Finally, Section 4.3 explains the identified use cases.

4.1 Problem Definition

Modeling and simulation tools have become more prevalent in industrial application development [19]. These tools facilitate various activities in the modeling and simulation lifecycle, such as defining requirements, creating, simulating, and checking models. Ensuring consistency among models in a multi-tool and multidisciplinary environment can be a challenging task. In this project, we propose a model management tool that extracts relationships among models and stores them using a graph database, which can be used for more insights and identifying inconsistencies among dependent models.

4.2 Project Goal and Scope

This project aims to contribute towards solutions to the design and development of a model management tool using a graph database. There are several sub-goals (high and low levels) in this project. The followings are the goals and contributions:

High-level goals:

- Identify inconsistencies and more insights in the early stage

Low-level goals:

- Design a model management tool with a graph database
    - We propose and design a model management tool. We check and identify the graph representation of a model, its elements, and its relationships. We also determine the elements of a model that should be represented in a graph.
- Design a graph data model (meta ontology) for a graph database
    - We define the graph data structure to be stored in the graph database. The meta-data makes the graph data of the model and the appropriate data required to define its relationship with other models.
- Design a parser for specific modeling tools to extract multi-domain model information
- Define use cases
    - We investigate how to identify possible use cases for defining and developing the model management process.
- Create a user interface for data visualization

Moreover, during the initial project meetings with the stakeholders, we defined our scope to focus on only the block-based modeling tools. This is because these modeling tools work together and there is a chance to create inconsistent situations among the dependent models. Also, we can store and represent the model information from block-based modeling tools in a graph database. Each block or element of the model becomes a node in the graph database.

4.3 Use Case Scenarios

Finding appropriate use cases is a challenging task. To achieve our project goal, we chose the following use cases. We defined these use cases to narrow down and specify our project scope and objectives.

4.3.1. Mismatching Inputs/Outputs (Interfaces)

One model may be connected with another model and dependent on each other regarding inputs and outputs. We can call these inputs and outputs interfaces between the models. An interface mismatch indicates the differences in the formats and specifications of messages exchanged between two interfaces. For example, we are developing a project where we use Simulink and Rhapsody modeling tools together. We need to use a Simulink model to simulate a task from a Rhapsody model. When the Rhapsody model is using the Simulink model, we need to provide the input parameter from the Rhapsody model to the Simulink model and receive the output result from the Simulink model. In this case, there is a possibility to mismatch the data type of the provided input parameters and as well as receiving parameters. In this project, these mismatching interfaces are also known as model inconsistencies. The inconsistency can happen when:

- The expected input data type of one model element and the corresponding desired output data type of another are different. We define it as model element-level inconsistency.
- The expected input data type of one model and the corresponding desired output data type of another differ. We define it as model-level inconsistency.

Figure 1.1 in Section 1.1 shows an example of the interface dependencies between the two models. Here, one Rhapsody SysML model is using another model of Simulink. The interfaces are marked as circles by which these two multi-tool models are connected and pass information to each other. If the data type of the model-level interfaces is not matched, then we can call it a mismatching interface.

4.3.2. Change Propagation

Change is very common in any system at any time. In a model-based development, a change not only impacts a model but also impacts other models. It is also known as impact analysis. We define change propagation as the required changes in other model elements to ensure consistency after a particular element is changed in one model. It helps us to identify the impact of any changes in the current system. Additionally, we become concerned about the modification and inform the respective team about potential model inconsistencies.

4.3.3. Model Evolution

As new elements and relationships are implemented, the model evolves continuously. Model changes can happen for several reasons. The model can be changed as new requirements are introduced to a modeled system or existing one's updated. These recent changes can result in a difference between the existing and dependent models. It allows the model management system to see what it looks like at a particular time.

Figure 4.2 shows an example diagram of model change history using the time-based versioning concept in a graph database. In this concept, we need to use two types of nodes. One type of node represents the id of each model element to uniquely identify it in the system. Another type of node keeps the latest information with a timestamp for every change.

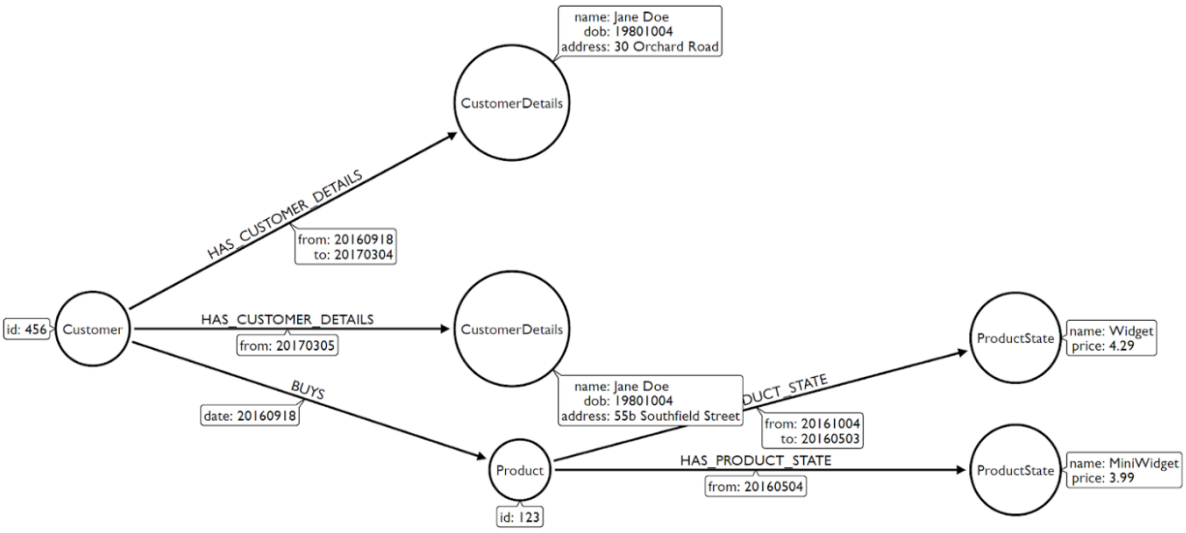

Figure 4. 1: Tracking model changes using time-based versioning in graph [24]

4.3.4. Input/Output Connectivity

We can check the input/output (interfaces) of model elements. It will help us to identify how a specific element is connected with other elements and what kind of data is exchanged between them. This way, we can analyze a model or its element in our modeling environment.

4.3.5. Hotspot Detection

In general, a hotspot can be defined as an area with a higher concentration of events than other events. In multi-tool modeling, the hotspot indicates which components have more than the usual importance or dependencies, or interconnections. It means which component has more incoming/outgoing connections. We can check hotspots in two ways:

- Model elements that have more than usual incoming/outgoing connections
- The model itself, which is used (by) many other models

4.3.6. Relationship Dependency

Relationship dependency indicates identifying the relationship between two elements or models. It will help us to see the full dependency path between the two elements, how they are connected, and which changes will impact them. Although it is similar to the input/output connectivity use case, it has a little difference. It will help us to check all possible relationship dependency paths between two specific model elements.

5.Requirement Analysis

This chapter explains the requirements that have been identified for this project. Section 5.1 describes the requirement overview. Section 5.2 provides an overview of the system's functional needs and how it is developed and prioritized. Finally, Section 5.3 shows the non-functional requirements of the system.

5.1 Requirement Overview

A requirement is a statement that defines a product or process's operational, functional, or design characteristic that is unambiguous, measurable, and necessary for its acceptance [25]. A clear list of requirements is essential to steering a project in the right direction. However, during the lifecycle of a project, new customer requirements can emerge, and the old ones may need to be updated.

This project has two categories of requirements: Functional and Non-Functional. Figure 5.1 describes these two categories. The functional requirement is related to the functionality that needs to be developed. On the other hand, the non-functional requirement is the requirement that defines the attributes of a system. Also, it directs the system's design criteria to satisfy these non-functional requirements.

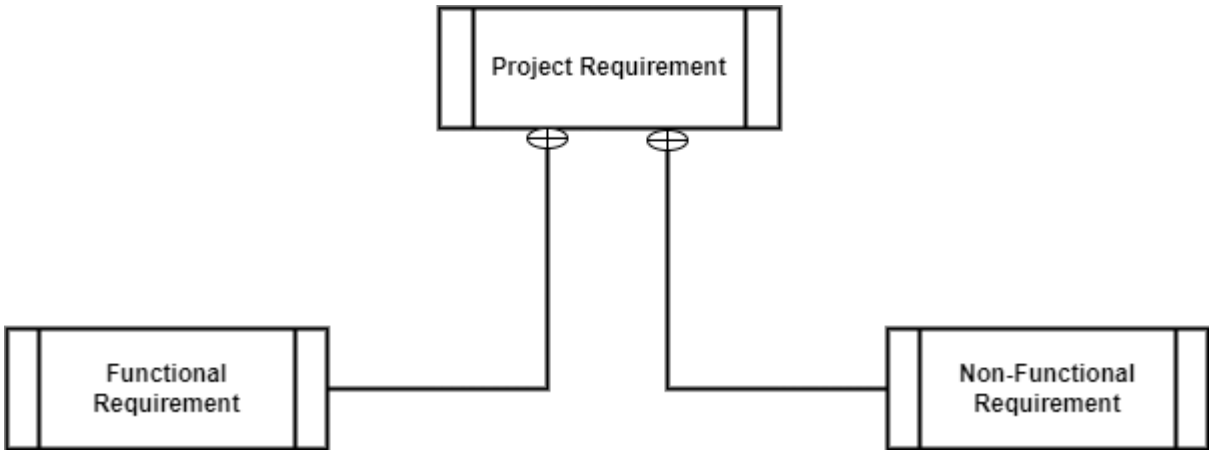

Figure 5. 1: Requirement Overview

We started with a set of initial requirements for this project. All of these requirements were revisited and updated throughout the entire project. The following techniques were used to acquire requirements:

- Brainstorming
- Stakeholder meeting
- Prototyping

The brainstorming session was held during the initial stage of the project. We started by analyzing the problem and possible solutions during the brainstorming session. After the session, we devised several functional requirements listed in the next section. Additionally, we arranged regular meetings with stakeholders to understand their needs. We developed prototypes to show and discuss our thoughts more actively to get early feedback.

The MoSCoW method [26] prioritizes the elicited set of requirements. The word MoSCoW is an abbreviation of four, each defining different priority levels. They are:

- **Must** have (M): The requirements under this category must be included in the final delivery.
- **Should** have (S): The requirements under this category are suggested to include in the project.

- **Could** have (C): The requirements under this category could be satisfied depending on the project's timeline.
- **Will** not have (W): The requirements under this category will not be addressed in the project scope.

5.2 Functional Requirements

In this section, we list the functional requirements. The project is broken down into several requirements, which are then decomposed into functions. The functional requirements are the requirements that describe a system as a specification of behavior between inputs and outputs. The various functional requirements are presented in Table 5.1.

Table 5. 1: Functional Requirements

Req_Id	Req_Priority	Req_Description
FR01	Must	The system shall allow the user to parse all models and extract data into the corresponding JSON file.
FR02	Must	The system shall allow the user to store extracted data in the graph database.
FR03	Must	The system shall identify all mismatching interfaces and allow the user to store them in a graph database.
FR04	Must	The system shall allow the user to visualize all mismatching interfaces existing in the current system.
FR05	Must	The system shall allow the user to visualize all mismatching interfaces existing in the current system.
FR06	Must	The system shall trace change propagation.
FR07	Could	The system shall detect model evolution. The system shall be able to store all relevant change information of a model. Also, the system shall allow the user to see the change history of a model.
FR08	Must	The system shall allow the user to investigate input/output connectivity.
FR09	Must	The system shall allow the user to detect the hotspot of a system.
FR10	Must	The system shall allow the user to find out relationship dependency.
FR11	Could	The system shall be able to read models from a git repository.
FR12	Should	The system shall allow the user to see all relevant visualization diagrams as required and expected.
FR13	Should	The system shall be able to create a graph data model for an individual modeling tool.
FR14	Should	The system shall be able to maintain a standard graph data model for all existing tools based on their model.

5.3 Non-Functional Requirements

The non-functional requirements (NFR) are criteria that assess a system's operation instead of its specific behavior. The various non-functional requirements are presented in Table 5.2.

Table 5. 2: Non-functional requirements

Req_Id	Req_Priority	Req_Description
NFR01	Must	The component of the system shall be decoupled and in a modular structure. The backend (graph database) must be separated from the frontend (visualization). Any changes in the frontend implementation must not affect the backend.
NFR02	Must	The system must be extensible to add new modeling tools on demand.
NFR03	Should	The system shall be user-friendly. The system shall use regular and appropriate user interface elements (e.g., bar graph, pie chart, table, etc.) so the user can easily understand them.
NFR04	Could	The system shall be testable with an automatic testing standard to make the testing process more manageable and effective.

6.System Architecture and Design

Proper planning is essential to implement a software system that satisfies the stakeholders' requirements. System architecture and design is one of the most important parts of that plan. A system architecture is a comprehensive description of the individual components that communicate and work together to make up the corresponding system.

This chapter describes the architecture that guided the development of the system. In Section 6.1, a high-level context and system architecture are described. Section 6.2 contains the 4+1 view model of architecture.

6.1 System Context

In this section, we illustrate the high-level context of the system, which explains the overall scenario before going to the detailed architecture. The system context diagram in Figure 6.1 is used to show the system's architecture. First, we store block-based models of different modeling tools in local storage. Next, our tool reads these models from the storage, extracts model information, and stores these in a graph database. It also creates a specific meta ontology (graph data model) for each modeling tool. Finally, we visualize data in the dashboard for identifying more insights.

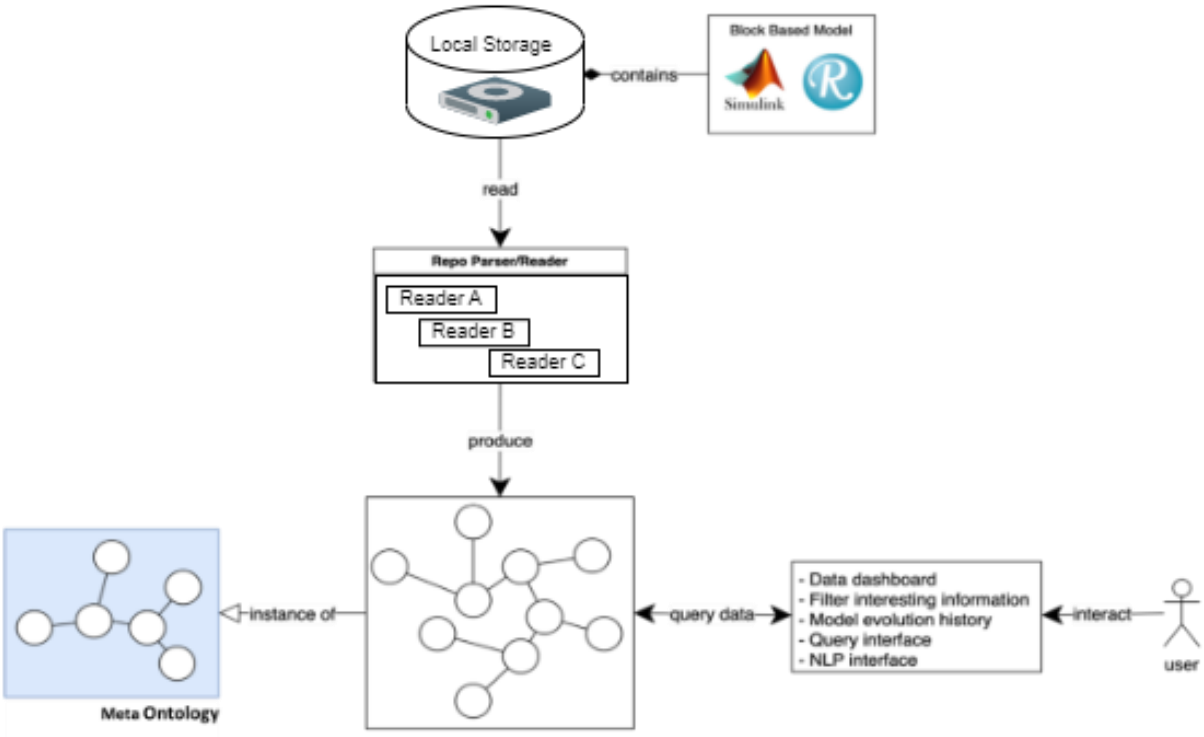

Figure 6. 1: High-level Context diagram of our system

The architecture of the entire system, based on the function decomposition, is presented in Figure 6.2. The system contains different components to extract model data, load data into the database and visualize data into the dashboard.

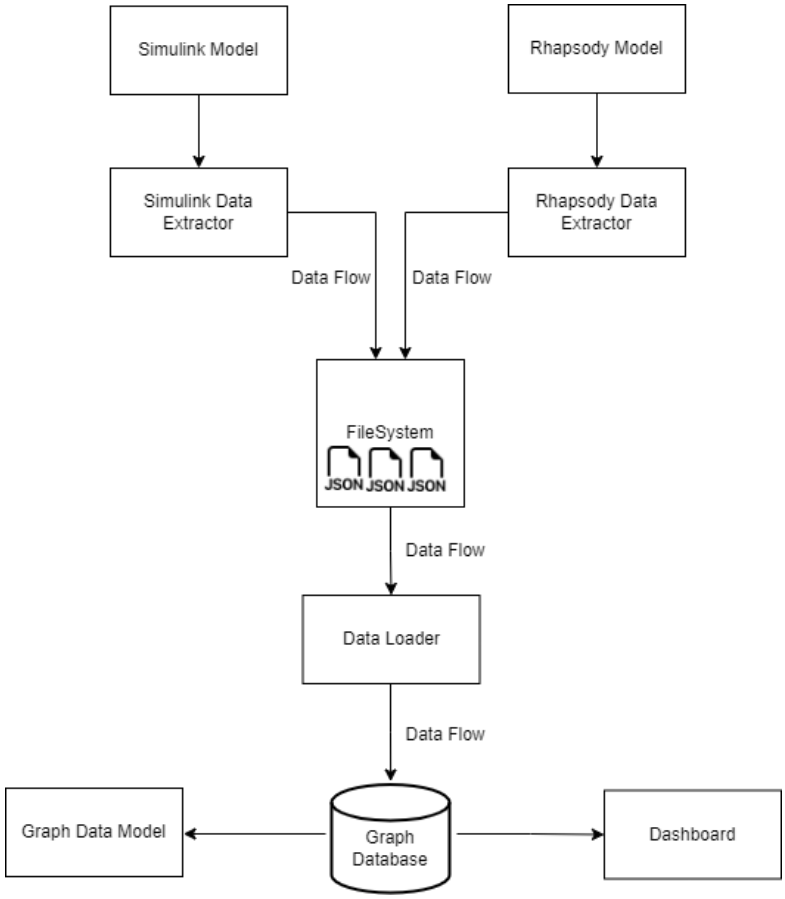

Figure 6. 2: Overview of the System Architecture

6.2 4 + 1 Architectural View

The 4+1 view model is one of the well-known and widely used architectural approaches for software-intensive systems proposed by Philippe Kruchten [34]. It has multiple views to describe separately the concerns of the various stakeholders for example system engineers, developers, and end-user. Therefore, this project used the 4+1 architectural view to illustrate different perspectives on describing the system.

The 4+1 architecture consists of four different views, which are:

- Logical view – describes the component (object) of the system and the interaction. In the UML diagram, this view can be demonstrated using a class diagram or state diagram.
- Process view – shows the processes of the system. This view can be illustrated using a sequence or activity diagram.
- Development view – illustrates a system from a programmer's perspective and is concerned with software management. UML component and package diagrams can be used to explain this view.
- Physical view – describes the installation, configuration, and deployment of the system. A UML deployment diagram can be used to illustrate this view.

The *one* from 4+1 architecture is Scenario. This is the fifth view, which represents the use cases that are supported by the system.

6.2.1. Use Case

We show use cases to describe potential interactions between a system and its users. In the context of this project, an admin designer is a typical user of the system. Figure 6.2 shows the potential actions that a user can perform with this tool.

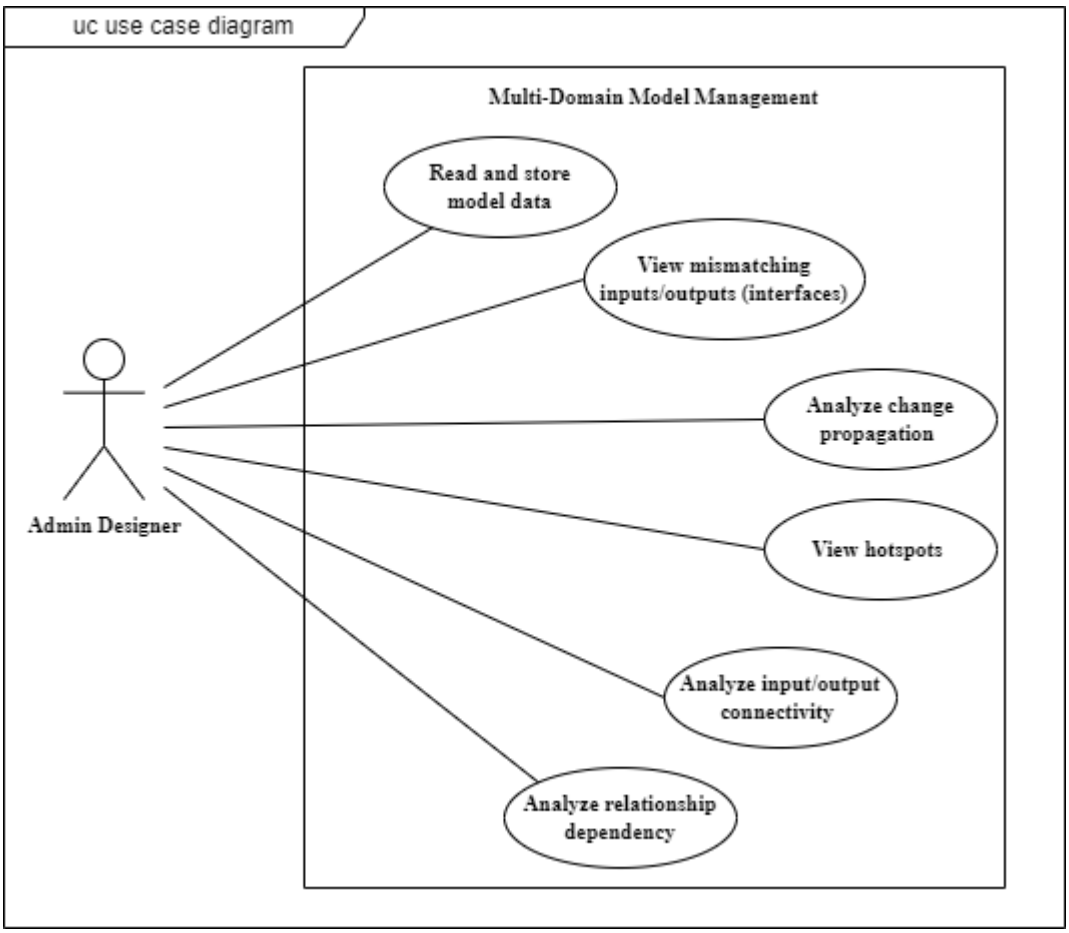

Figure 6. 3: Use cases diagram of the system

6.2.2. Logical View

This section describes the logical view of the system. The logical view is concerned with the system's functionality to end-users. We are using the data extractor and loader class diagram for this view, as Figure 6.3 shows.

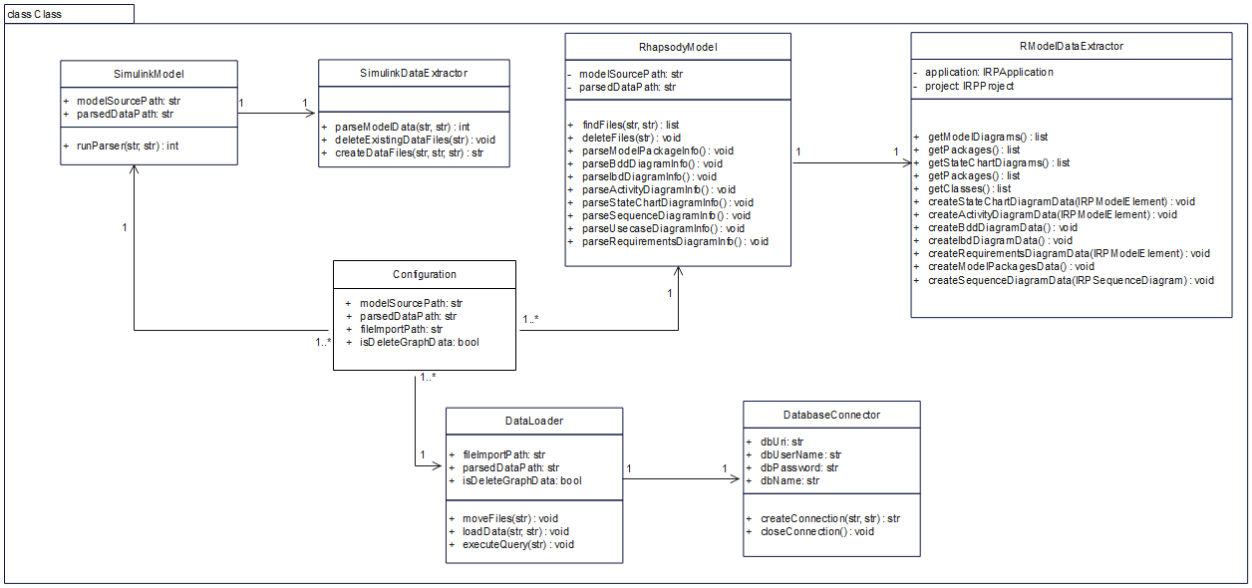

Figure 6. 4: Data extractor and loader class diagram

6.2.3. Process View

This process view includes the UML sequence diagram of the data visualization. Sequence diagrams are used to illustrate how classes behave and interact to accomplish a specific use case functionality. It is often easier to understand the dynamic behavior of a given system's use case processes by the sequence diagram. Figure 6.5 shows the sequence diagram for the admin designer. Initialization would start from the admin. The sequence is broken into steps and listed below:

- The admin requested the GUI module to view the data
- The GUI selects specific data visualizer from the Data Visualizer by requesting data from the Database
- If the database contains the latest requested data, the response will be returned to the admin user through Data Visualizer and GUI module.

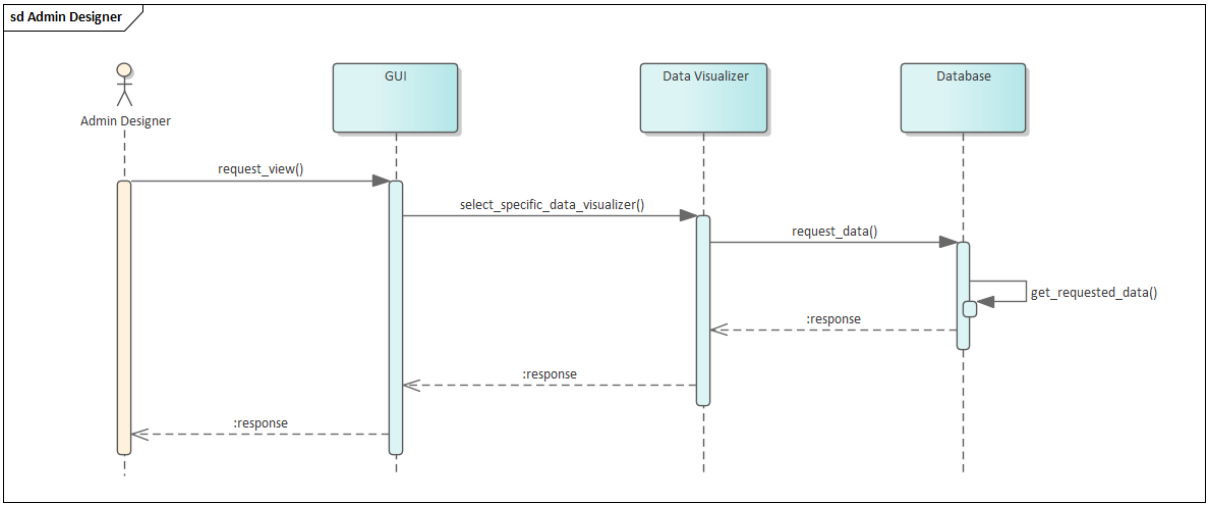

Figure 6. 5: Data visualization Sequence diagram

Figure 6.6 shows the sequence diagram for storing model data in a database. The sequence is broken into the steps below:

- The admin runs the data generator to the Updated Data Generator module
- This module generates the updated data from the models

- The generated data is sent to the Database module to store
- The response will be sent to the Admin Designer if the data is stored successfully in the Database.

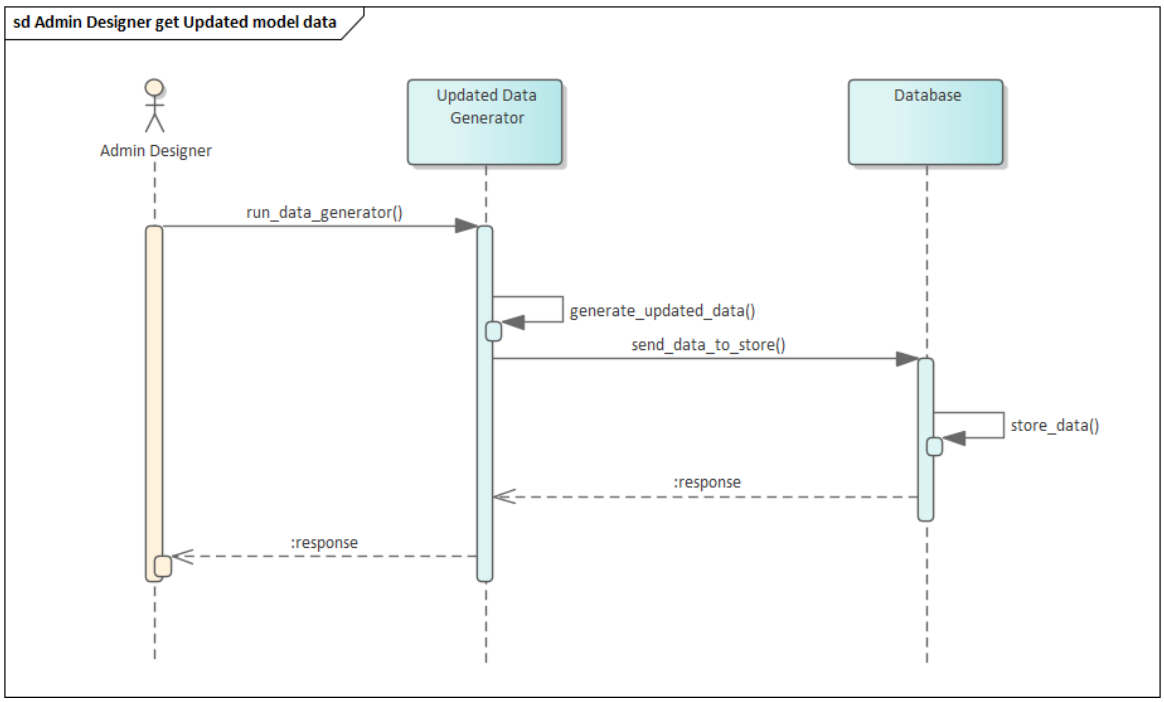

Figure 6. 6: Storing model data into database Sequence Diagram

6.2.4. Development View

The development view, also known as the implementation view, investigates a system from a programmer's perspective and is concerned with software management. The component diagram described in Figure 6.6 is used to depict the development view.

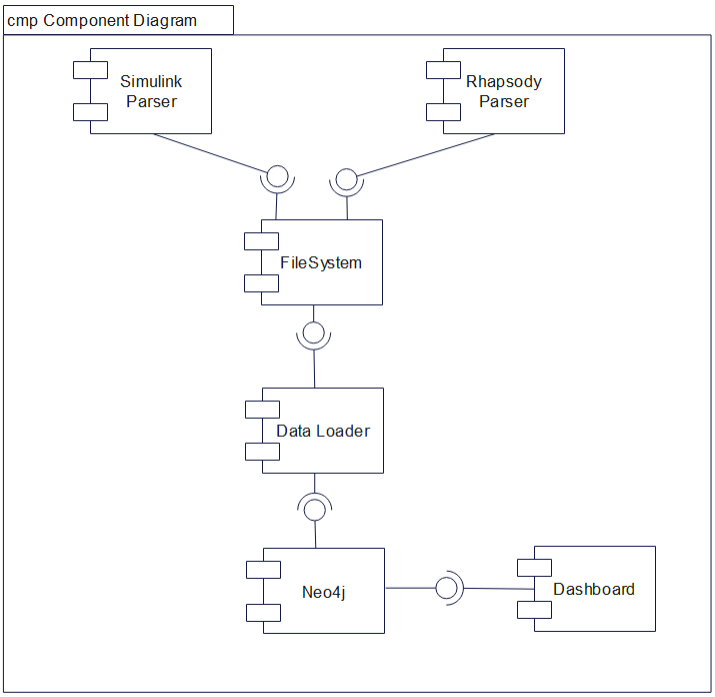

Figure 6. 7: System component diagram

6.2.5. Physical View

The physical view, also known as the deployment view, depicts the system from a system engineer's point of view. Figure 6.7 presents a visual overview of the deployment of various entities in the system. It shows us the different components of the system.

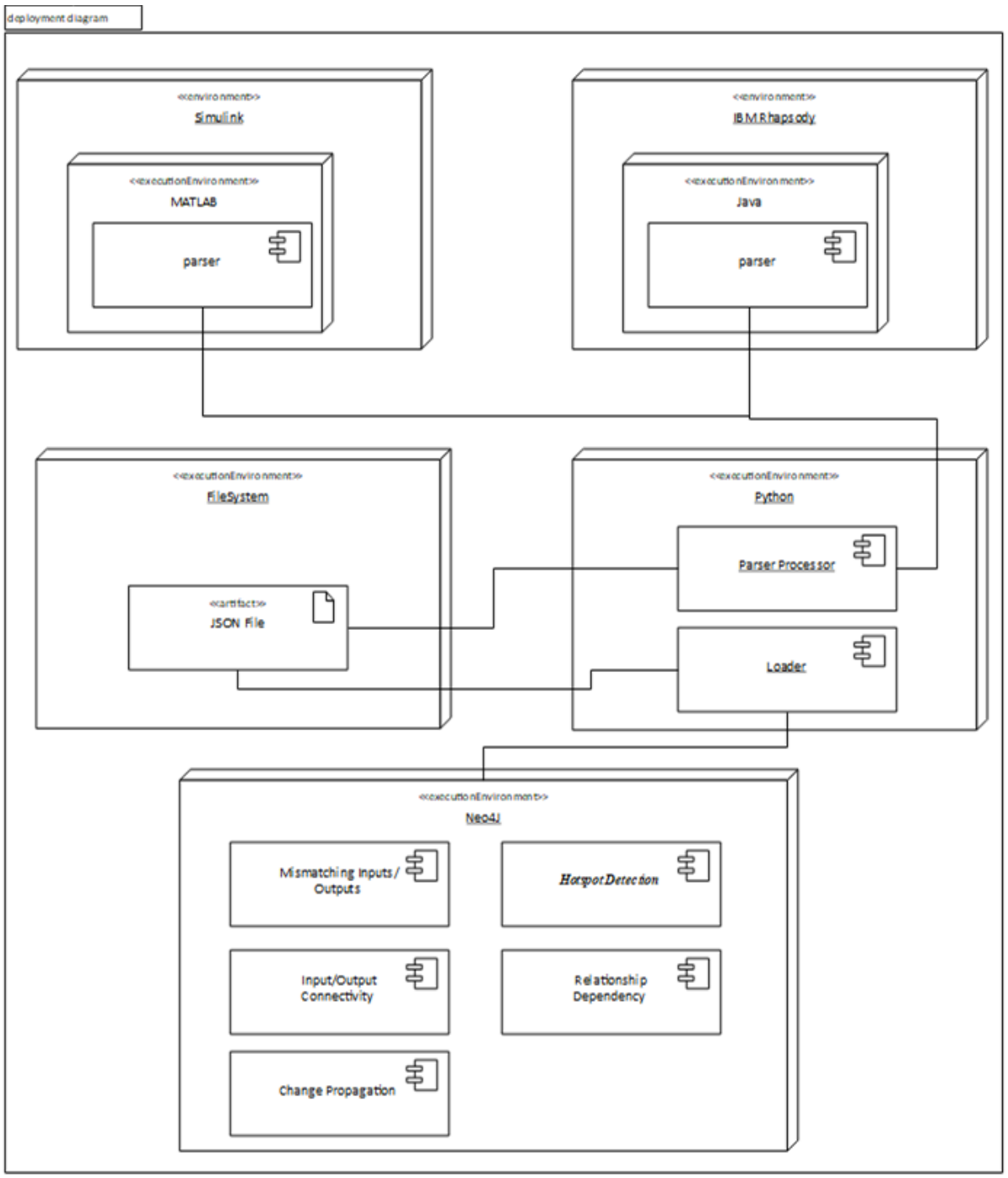

Figure 6. 8: System deployment diagram

7.Implementation

This chapter describes the implementation of the system. In Section 7.1, we explain the choices for selecting tools and technologies to develop this project. Afterward, in Section 7.2, we describe the Simulink model implementation in more detail. Also, we explain the Rhapsody SysML model implementation in Section 7.3. We describe a combined graph data model in Section 7.4. Finally, we explain data visualization in Section 7.5.

7.1 Technology Choice

The project was developed according to the architecture explained in Chapter 6. We choose several technologies to implement this graph data modeling tool. This section explains the choices for selecting tools and technologies to develop this project.

7.1.1. System Requirements

To develop the project, we use the following software packages.

- **MATLAB & Simulink** → Version R2021a
- **IBM Rhapsody** → Version 9.0.1
- **Eclipse IDE** → Version 2022-06 (4.24.0)
- **Neo4j Desktop (Database)** → Version 1.4.15
- **Python** → Version 3.7
- **PyCharm (Professional Edition)** → Version 2021.3

7.1.2. Database Selection

We were looking for a database, especially a graph database. We compared and investigated several databases to select one for our project. Table 7.1 shows a comparison of different graph databases with some criteria.

Table 7. 1: Database selection comparison [12]

Database	Graph Data Model	Query Language	Open Source
ArangoDB	multi-model	ArangoDB Query Language	community edition
AllegroGraph	RDF	SPARQL, Prolog	no
InfiniteGraph	Property Graph Model	"DO"	no
OrientDB	multi-model	Gremlin, SQL	yes
Neo4j	Property Graph Model	Cypher	community edition

Although there were slight differences between the databases, we have chosen Neo4j as the graph database for this project. Following are the overall selection points of Neo4j:

- NoSQL Graph Database
- Open-source
- Enables ACID-compliant transactions
- Supporting a friendly query language called Cypher
- Offers better performance in retrieving data
- Ease of installation and use

- Largest graph data community
- Provides an easy way to utilize APIs, extensive libraries
- Exist available tools that are used to operate the data and visualize the stored graph

7.1.3. Visualization tool Selection

We explored existing dashboard tools for the Neo4j database. Based on Appendix A, we see that NeoDash is a reporting tool developed as a community project. This tool has features such as:

- An open-source, low-code dashboard builder developed by Neo4j Labs
- Drag-and-drop interfaces
- Ability to add customization and interactive dashboard options
- Create visualizations directly from Cypher query
- Support different data presentation options e.g., tables, graphs, bar charts, maps, and more
- Save dashboards to the database and share them with others
- Build and publish dashboards for read-only access

Based on these features, we have chosen NeoDash to visualize data in this project. Figure 7.1 shows an example dashboard developed in the NeoDash tool.

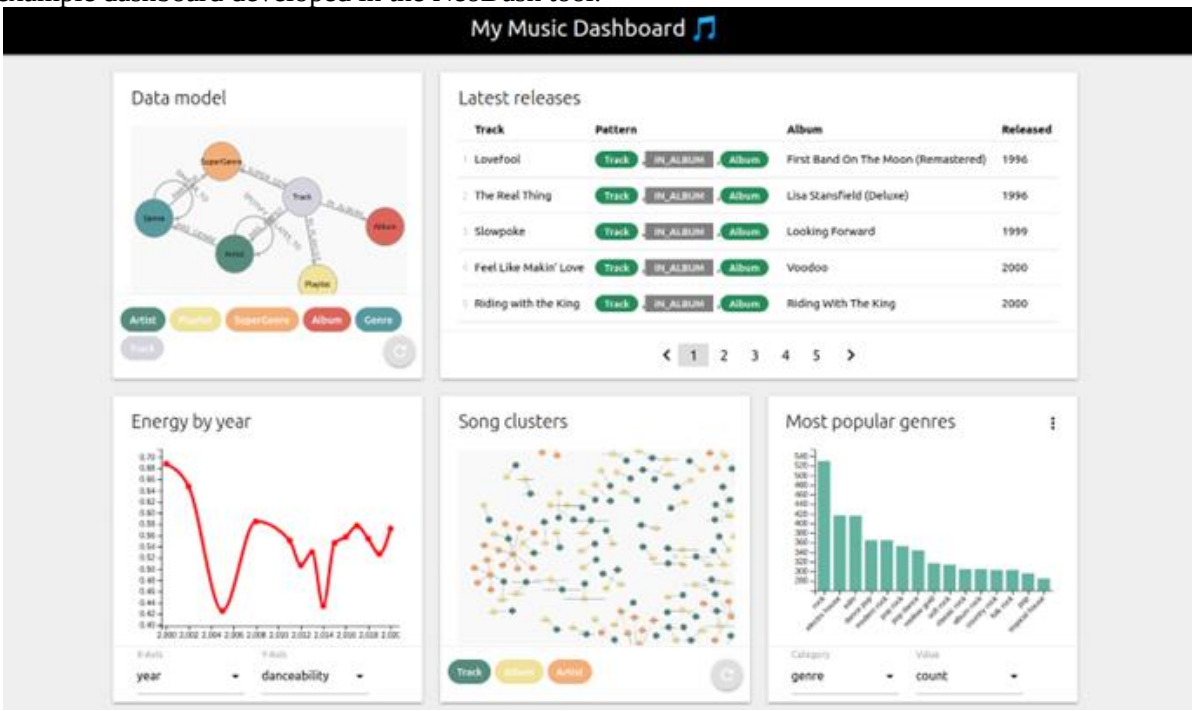

Figure 7. 1: An example dashboard developed in NeoDash

7.2 Simulink Model Implementation

In this section, we describe the details implementation process of the Simulink model.

7.2.1. Simulink Model

In Simulink, a model is a collection of blocks that represents a system. These blocks are connected with lines. We can develop a Simulink model with different levels of complexity. Figure 7.2 shows an example Simulink model with blocks. This model also has subsystem blocks. A subsystem block looks like an inner model (model inside a model).

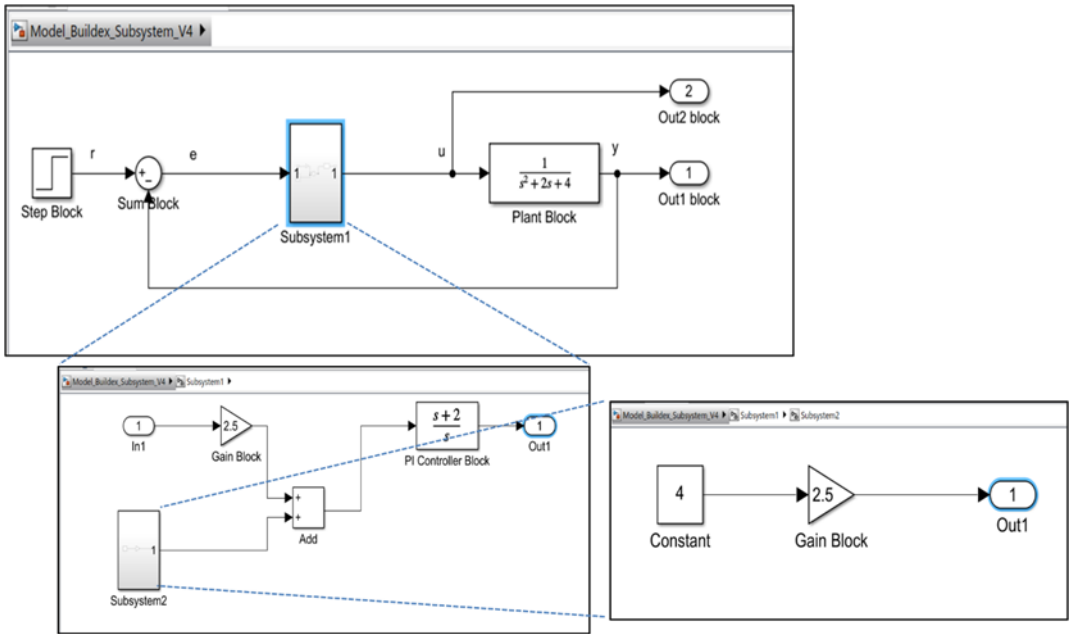

Figure 7. 2: Simulink example model

7.2.2. Simulink Parser Workflow

The translation of Simulink models into the graph is implemented using MATLAB-script with MATLAB API and Python scripts described in Figure 7.3. This figure provides an overview of the implementation. The retrieve_Simulink_model.py script reads the Simulink model and extracts blocks, relationships, and their properties information. The extracted information is stored in JSON files. We generate one JSON file for each Simulink model. This enables a separation of concerns for each model and enables testing the retrieve functions. The insert_Simulink_Model_in_Neo4j.py script load the stored JSON files and writes every component via a separate query into the Neo4j database. Finally, we visualize graph data in a dashboard.

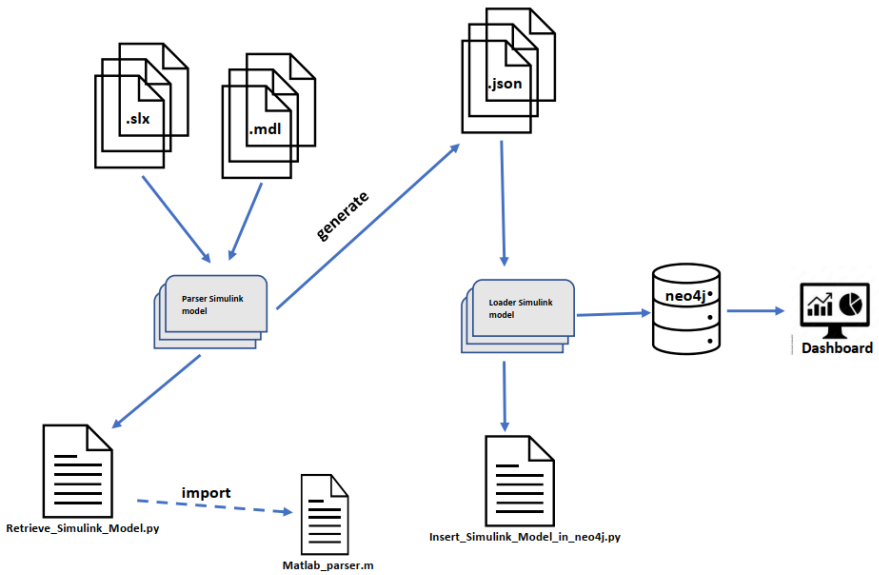

Figure 7. 3: Simulink parser workflow

Figure 7.4 describes the algorithm for parsing Simulink models.

Algorithm 1 Parse Simulink Model Data

INPUT: source directory of all slx or mdl model files
OUTPUT:json data file for each model

```
prepare the list for all slx or mdl model files
for each model file in the list do
    create a node that represents the model
    prepare the list for all blocks in the model
    for each block in the list do
        create a node that represents the block
        create a relationship that connects the block with the model
    end for
    prepare the list for all ports in the model
    for each port in the list do
        create a node that represents the port
        create a relationship that connects the port with the corresponding
        block
    end for
    prepare the list for all lines/signals in the model
    for each line in the list do
        create a relationship that connects the line with two corresponding
        port nodes
    end for
    create a json file for each model
end for
```

Figure 7. 4: Simulink parser algorithm

7.2.3. Simulink Graph Data Model

After analyzing a couple of Simulink models and based on the literature review, we created a graph data model to represent any Simulink model. This graph data model represents the nodes as Simulink model elements and the relationships as how these elements are connected. It also shows properties of each node and relationship. We developed our data extraction process according to this graph data model. Figure 7.5 represents the graph data model of Simulink models.

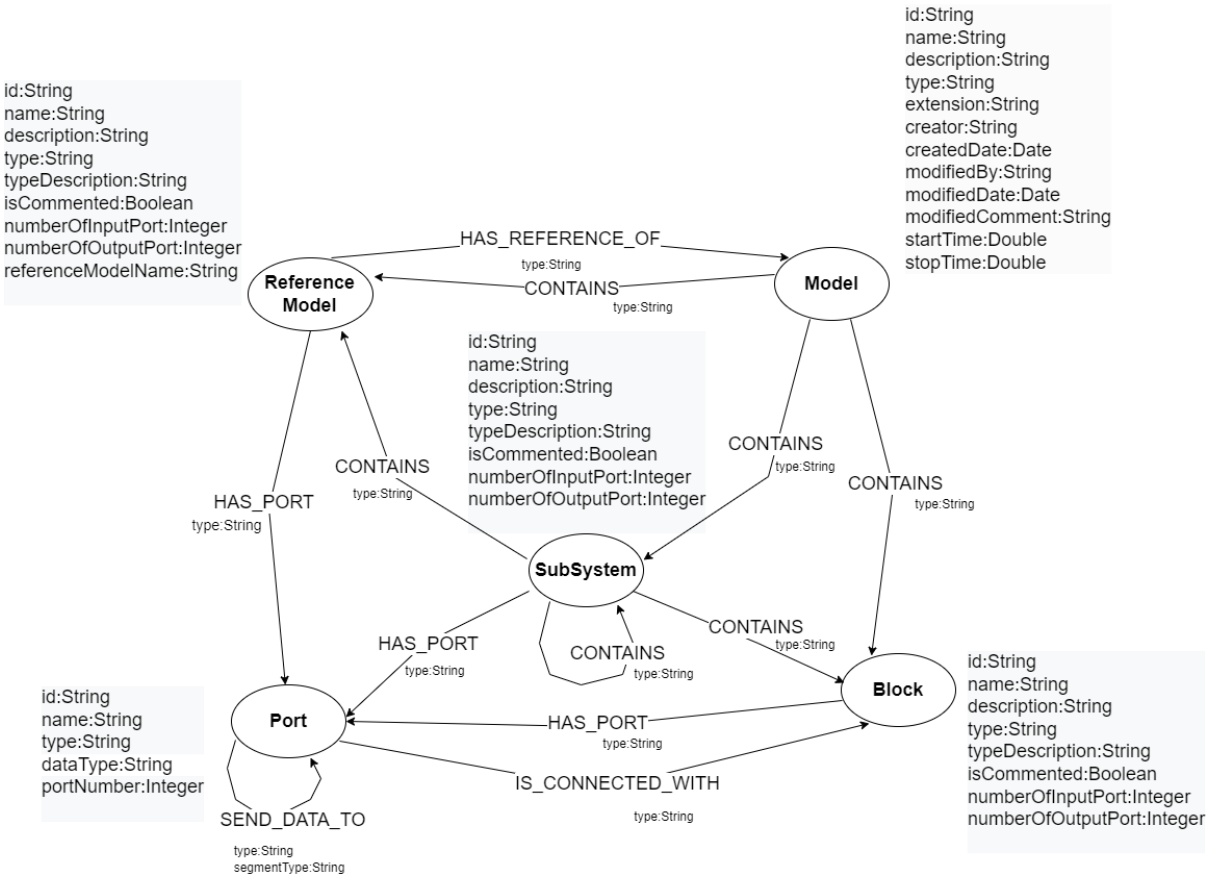

Figure 7. 5: Simulink Graph Data Model with entities and their properties

7.2.4. Simulink Graph Data

We see the graph after loading the extracted JSON data into the database. Figure 7.6 describes the graph for the model shown in Figure 7.2.

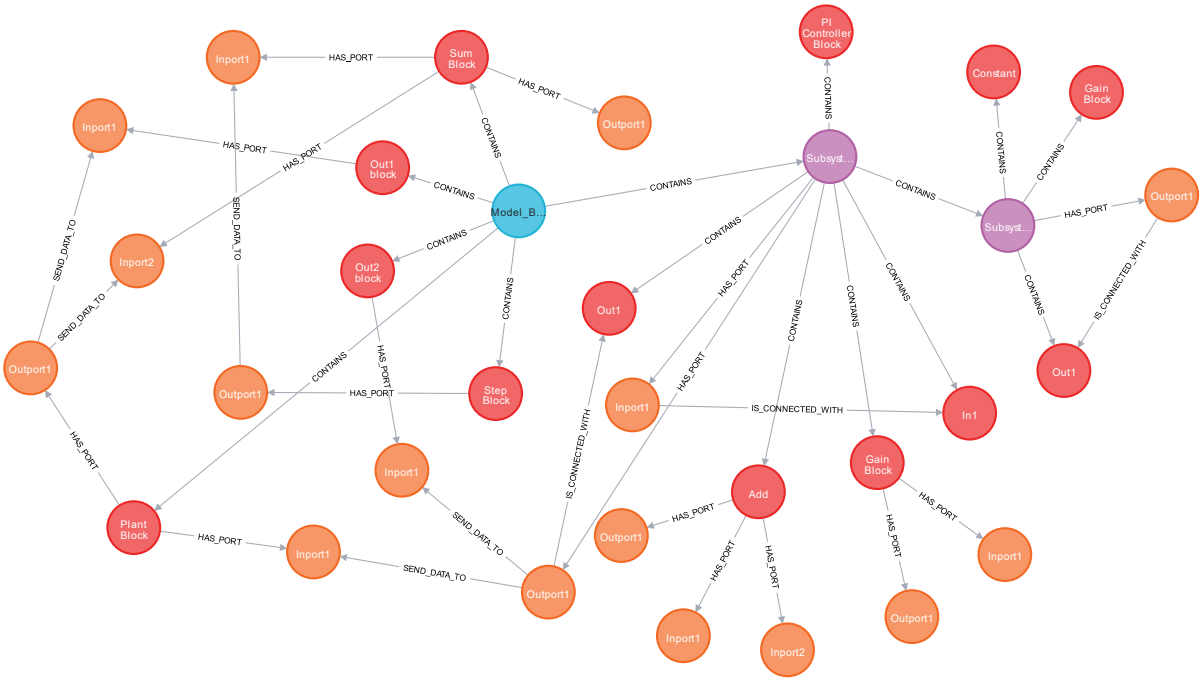

Figure 7. 6: Generated graph from the Simulink model

7.3 Rhapsody SysML Model Implementation

In this section, we describe the details implementation process of Rhapsody SysML models.

7.3.1. Rhapsody Model

Rhapsody SysML has many diagrams according to Section 3.2.2. Figure 7.7 shows an example block definition diagram.

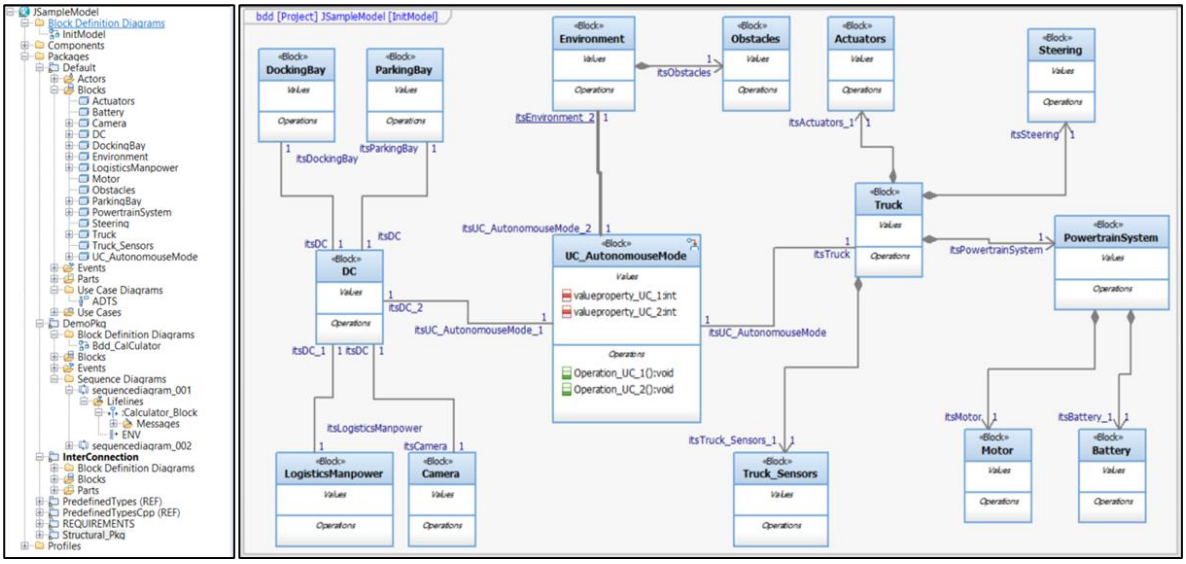

Figure 7. 7: Rhapsody Model Structure with Block Definition Diagram

7.3.2. Rhapsody Parser Workflow

Figure 7.8 provides an overview of the Rhapsody parser implementation. The system reads the SysML models generated in Rhapsody and extracts SysML model elements and relations. We generate one JSON file for each Rhapsody SysML diagram of a model.

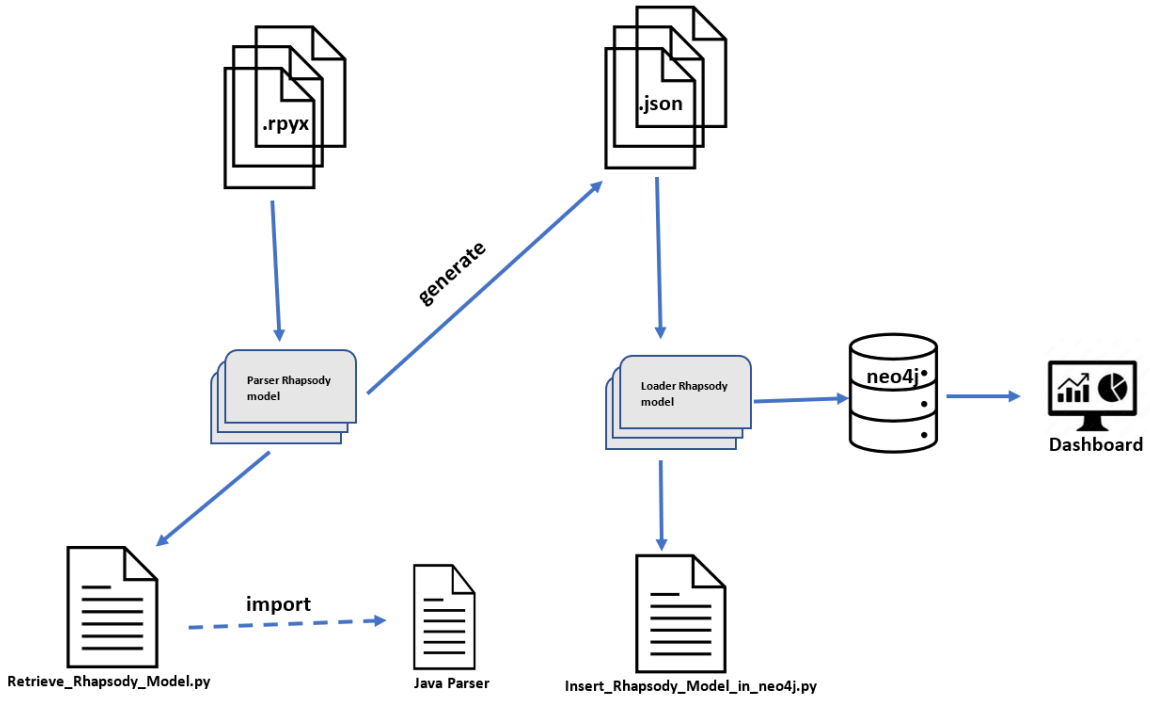

Figure 7. 8: Rhapsody parser workflow

Figure 7.9 describes the algorithm for parsing Rhapsody models.

Algorithm 1 Parse Rhapsody Model Data

INPUT: source directory of all rpyx files
OUTPUT:json data file for each model

```
prepare the list for all rpyx model files
for each model file in the list do
    create a node that represents the Project
    prepare the list for all packages in the model and create a node for
    each packages
    create a json file for each model and corresponsing packages
    prepare the list for all specified types of diagram in the model
    for each type of diagram in the list do
        prepare the list for all diagrams of this type in the model
        for each diagram in the list do
            create a node that represents the Diagram
            prepare the list for all elements in the diagram
            for each element in the list do
                create a node that represents the element
                create a relationship that represents the element
            end for
            create a json file for each diagram
        end for
    end for
end for
```

Figure 7. 9: Rhapsody parser algorithm

7.3.3. Rhapsody Graph Data Model

We created a graph data model to represent any Rhapsody model based on the extracted model information. This graph data model represents the nodes as Rhapsody model elements and the relationships as how these elements are connected. It also shows properties of each node and relationship. As like as Simulink model, we also developed our data extraction process according to this graph data model. Figure 7.10 shows the graph data model of Rhapsody SysML models.

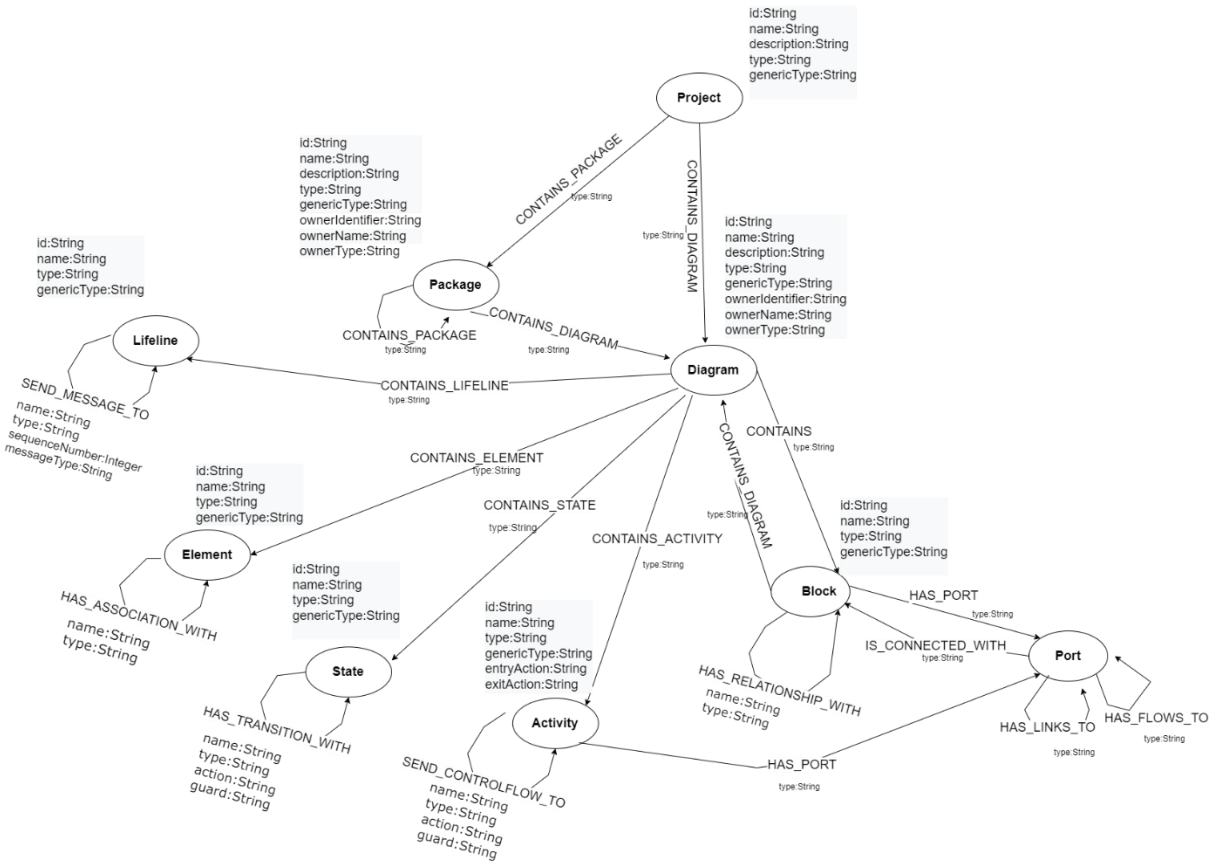

Figure 7. 10: Rhapsody Graph Data Model with entities and their properties

7.3.4. Rhapsody Graph Data

Figure 7.11 describes the graph for the model shown in Figure 7.7 after loading the extracted JSON data into the database.

Figure 7. 11: Generated graph from the Rhapsody model

7.4 Combined Graph Data Model

When we want to analyze multiple modeling tools models with a tool, we need to store these tools' data in a common graph database. It means that we need to create a combined graph data model of these modeling tools. This common graph data model is necessary to represent all the models and their internal relationships with different tools. We ignored the properties of each entities for the simplicity.

In this project, we developed an individual graph data model for Simulink and Rhapsody models based on Subsection 7.2.3 and 7.3.3 respectively. We generated a combined graph data model from these two models. Figure 7.12 shows the fusion of graph data models for Simulink and Rhapsody models.

Figure 7. 12: Combined graph data model with only entities

7.5 Data Visualization

We developed a dashboard to visualize data using the NeoDash dashboard tool. This dashboard has separate pages (tabs) to show each use case based on model data stored in the Neo4j graph database. This dashboard helps to query and analyze for more insights on model data stored in the Neo4j database. It also helps to identify inconsistencies among dependent models. Figure 7.13 shows an overview page of dashboard visualization. We can go to several pages (tabs) to see and analyze several use cases as we described in Section 6.2.1.

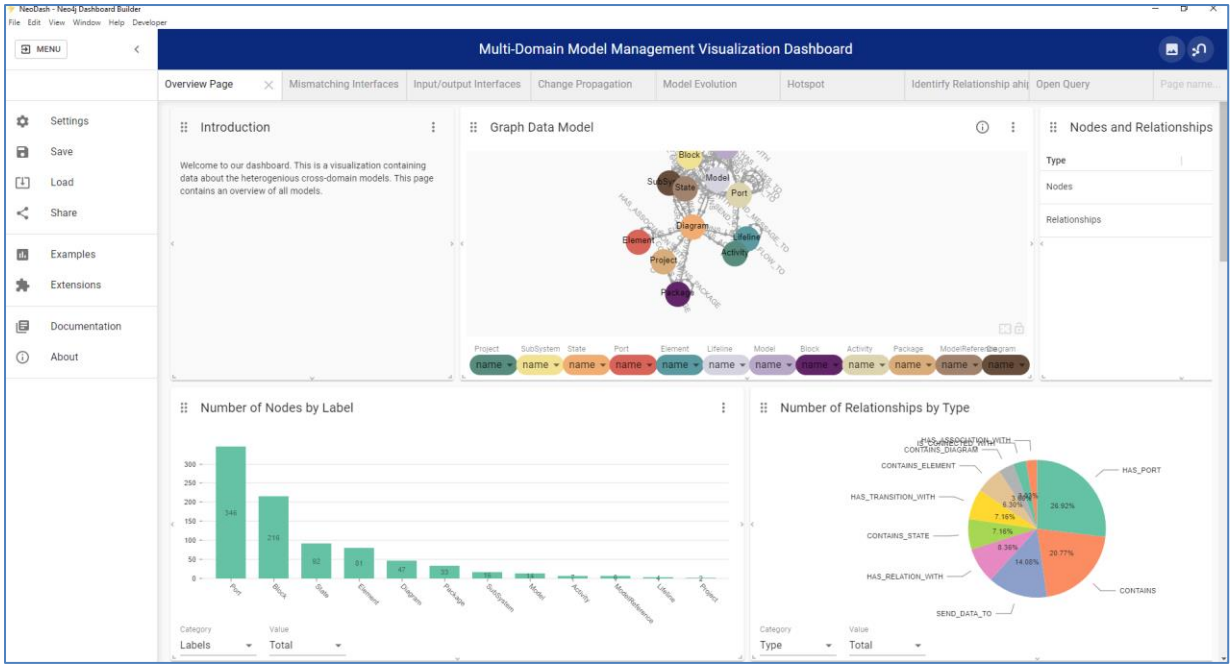

Figure 7. 13: Dashboard to visualize data

8.Verification & Validation

Verification and validation are vital steps in a software/system development process. Verification evaluates whether the system is implemented well, whereas validation evaluates whether the system meets the needs of the stakeholders [27]. In this chapter, we will describe the process of verification and validation we used in this project and the results. As a result, the verification and validation processes were employed to ensure that the tool implementation met the project's requirements.

8.1 Verification

Verification is usually an internal process of evaluating the correctness of a system's development and implementation process. For the verification process, we performed two ways: unit testing and integration testing. Each of these processes is briefly described in the following sections.

8.1.1. Unit Testing

Unit test is a software testing method that checks the individual units of the corresponding software have expected behavior [28]. A software unit can be a function of a class. The developer typically performs these tests by writing additional code that automatically tests the software. In the context of this project, unit tests were used not only to test the newly implemented features but also to ensure that existing functionalities were not broken. We developed test cases for every component. Some of these test cases for the system are listed in Table 8.1.

Table 8. 1: Existing sample test cases for the project

Test Case Id	Test Name	Description	Expected Results	Status
TU01	Check destination folder was cleaned before generating new data files	• Run model parser • Check the destination folder before creating the data file • All the existing files will be deleted from the folder	All the existing files will be deleted from the folder.	Passed
TU02	Check all model files are identified to read by the model parser in the repository	• Run Simulink parser • Check the number of files identified in the project folder as Simulink model by Simulink parser • Check the number of files in the folder	File count by script and file count by manual will be matched	Passed
TU03	Check a given model file is extracted and generated data file	• Run Simulink parser • Check a JSON data file created for a specific model file	The data file will be created for a specific model file	Passed
TU04	Check all model files are parsed and generated corresponding data files	• Run Simulink parser • Check all JSON data file has been generated for this model	JSON data files will be generated with relevant model files	Passed

TU05	Check a generated data file contains all elements of the model file	• Run Simulink parser • Check a JSON data file created for a specific model file • Check the elements in the data file	Elements in the data file will match elements in the model file	Passed
TU06	Check database connection with valid data to load model information	• Run data loader before loading data • Enter database name • Enter database password • Check the connection with the database	The connection will be established successfully	Passed
TU07	Test database connection with invalid data to load model information	• Run data loader before loading data • Enter an invalid database name • Enter a valid database password • Check the connection with the database	The connection will not be established	Passed
TU08	Test the same data loaded successfully into the database	• Run data load process • Check the query to see the expected data loaded into the database	The query will return the same data according to the data file	Passed
TU09	Check frontend dashboard and backend database connection with valid data	• Open the NeoDash dashboard tool • Enter the database project name • Enter database name • Enter database password • Check the relation with the database	The connection will be established with the database successfully	Passed
TU10	Check the frontend dashboard and backend database connection with invalid data	• Open the NeoDash dashboard tool • Enter the database project name • Enter the valid database name • Enter invalid database password • Check the connection with the database	The connection will not be established and shown a login error message	Passed
TU11	Test graph data model is generated into the database	• Insert data into the database • Open the Neo4j query window	The graph data model will be shown	Passed

		• Run the query to check schema visualization • Check the schema		
TU12	The test graph data model is updated based on loaded data	• Insert new modeling tools data into the database • Open the Neo4j query window • Run the query to check schema visualization • Check the schema is updated based on the new tool	The latest graph data model will be shown	Passed
TU13	Test graph data is visualized in the dashboard	• Open the NeoDash dashboard tool • Open a tab to visualize hotspot detection • See the model element with the highest relationship value • Check the same element's name and count it into our model	Element name and relationship count will be the same in both cases	Passed

Figure 8.1 shows unit test case status results for Simulink models in MATLAB script. These tests verified the implementation of the code for parsing Simulink model files. We used the MATLAB unit testing framework to write and run unit tests and analyze test results.

```
>> simulinkTests = matlab.unittest.TestSuite.fromClass(?simulinkUnitTest);
>> result = run(simulinkTests);
>> simulinkTests = matlab.unittest.TestSuite.fromClass(?simulinkUnitTest);
>> result = run(simulinkTests);
Running simulinkUnitTest

Done simulinkUnitTest
_________

>> rt = table(result)

rt =

  5×6 table

                                  Name                                  Passed    Failed    Incomplete    Duration      Details
    ____________________________________________________________________    ______    ______    __________    _________    _________

    {'simulinkUnitTest/checkDestinationFolderCleaned'                  }    true      false       false       0.0043927    {1×1 struct}
    {'simulinkUnitTest/checkAllModelFilesIdentified'                   }    true      false       false       0.0041048    {1×1 struct}
    {'simulinkUnitTest/checkSpecificDataFilesCreated'                  }    true      false       false         0.15407    {1×1 struct}
    {'simulinkUnitTest/checkAllDataFilesCreated'                       }    true      false       false       0.0041027    {1×1 struct}
    {'simulinkUnitTest/checkDataFilesContainsExpectedElements'}    true      false       false       0.0034617    {1×1 struct}
```

Figure 8. 1: The unit testing status result of the Simulink models implemented in MATLAB

For unit testing of our Python implementation, we used the PyTest library. Our decision to use PyTest was based on its maturity and comprehensive documentation. Figure 8.2 shows the results of the unit test cases implemented in PyTest.

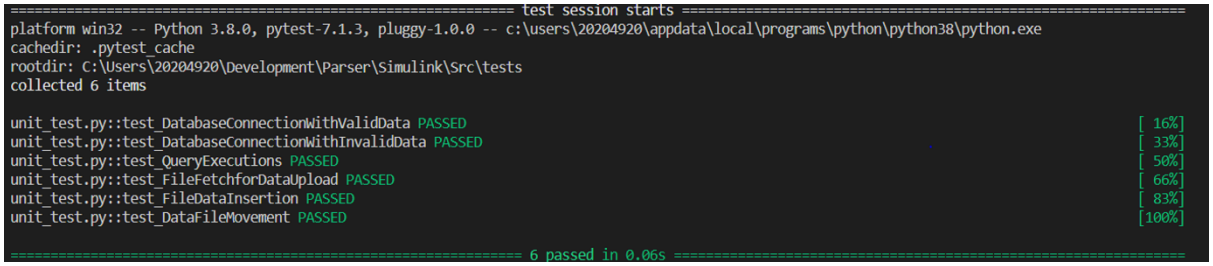

Figure 8. 2: Unit test status implemented in Pytest

8.1.2. Integration Testing

Integration tests are performed to determine if the units are working as expected after integrating them. This testing is performed after the unit testing. The main goal of this testing is to check the interfaces between the components. In the context of this project, we developed the backend and frontend separately. For the backend development, we developed parsers for different design tools. Then we integrated all these individual developments to make a complete backend system.

Moreover, we developed a parser of the Simulink model in MATLAB, which generated JSON data files for each model. But we developed scripts in Python to upload data from JSON files to the Neo4j database. Therefore, we checked the collaborated performance of system components developed in different programming languages through integration testing. Using the specification as a guide, we developed the frontend and integrated it with the backend. These tests indicated whether the integration was working correctly. Table 8.2 shows the manual test cases for checking the integration of different components.

Table 8. 2: Manual test cases for checking the integration of different components

Test Case Id	Test Name	Description	Expected Results	Status
TI01	Check Simulink JSON files generated and loaded	• Run Simulink parser with a model • Check a JSON file has been generated • Check the data file loaded into the database successfully	The Simulink model data file will be generated and loaded into the database	Passed
TI02	Check Rhapdosy JSON files generated and loaded	• Run the Rhapsody parser with a model • Check a JSON file has been generated • Check the data file loaded into the database successfully	The rhapsody model data file will be generated and loaded into the database	Passed
TU03	Check Simulink, and Rhapsody module works together	• Run Simulink and Rhapsody parser with a model • Check a JSON file has been generated for each corresponding model file	JSON data file will be generated for each model file together	Passed

TI04	Check the frontend dashboard and backend database connection	• Open the NeoDash dashboard tool • Enter the database project name • Enter database name • Enter database password • Check the connection with the database	The connection will be established with the database successfully	Passed

8.2 Validation

Validation is primarily an external process of evaluating a system or component at the end of the development process to determine whether it satisfies specified requirements [28]. This process aims to ensure that the right product has been developed and meets the stakeholder's expectations. In the context of this project, the verification process was carried out by the trainee and the key stakeholders (stakeholder analysis in Chapter 2) in different phases, which are discussed in detail in the following sections.

8.2.1. Regular Stakeholder Feedback

This project followed an incremental tool development process. Multiple weekly meetings were arranged with the key stakeholders. The purposes of these meetings were to keep the stakeholders involved and aligned in the development process, perform immediate validations, and identify varying requirements as early as possible. We used several architectural and design diagrams during these meetings to explain the implementation process. Based on these discussions, the stakeholders could identify whether the development activities were progressing in the right direction. The mentioned diagrams are presented and explained in Chapter 5 and Chapter 6.

8.2.2. Project Goal Evaluation

According to the project timeline and plan explained in Section 9.2, we set several milestones in the design and implementation phases. The system requirements and the tool implementation were discussed during the monthly PSG meetings to ensure that the project was on track. It also confirmed that the implemented system was built according to the agreed specifications. Moreover, several feedbacks were received during the demonstration that indicated the direction of this project.

9.Project Management

Project management is an essential aspect of any project's success. Each EngD ST project involves challenges not only technically but also organizationally. This project uses an iterative approach, allowing us to demonstrate the progress to the project client and get valuable feedback in the early stage.

This chapter reflects on the project management and planning of this project. First, we describe how we managed the project. Second, we show the possible identified risk of this project.

9.1 Work-Breakdown Structure (WBS)

In this section, the Work-Breakdown Structure of the project is discussed. We divided the project period into five phases: Planning and Management, Research, Design and Implementation, Validation and Verification, and Project Closure. Figure 9.1 shows the activities conducted in each step.

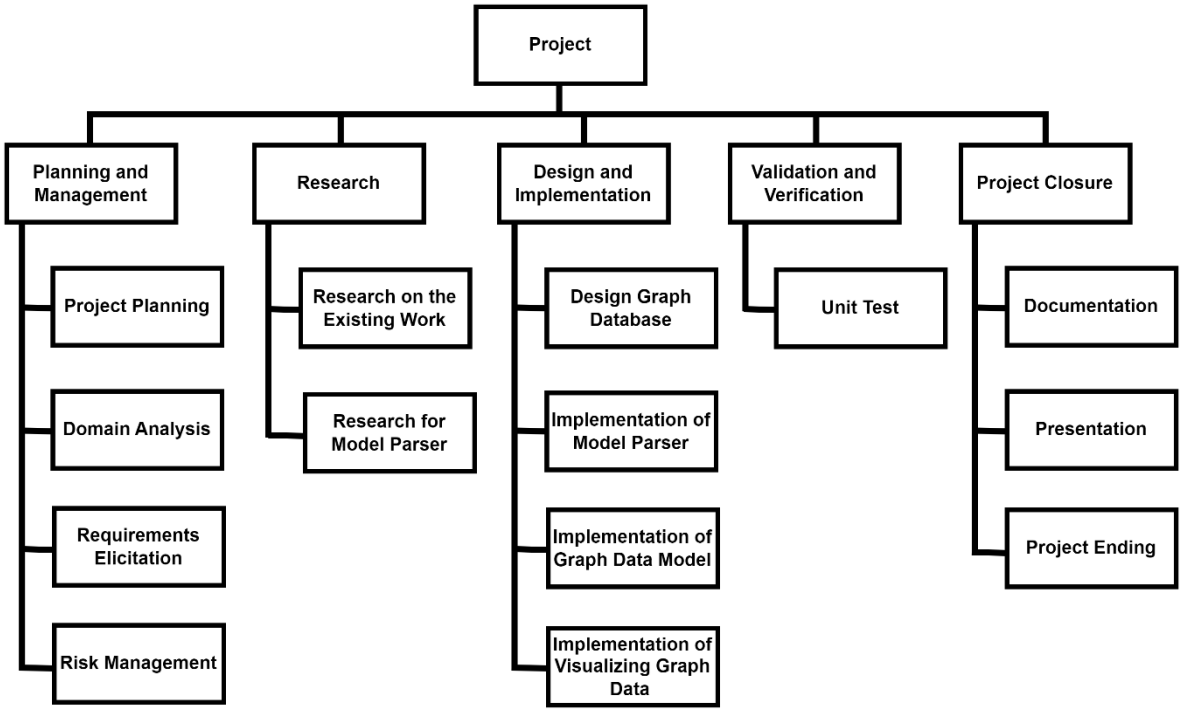

Figure 9. 1: Project Work-Breakdown Structure

9.2 Project Planning and Scheduling

At the beginning of the project, there were some questions about the requirements. We came up with the initial plan to start the project, and after each PSG meeting, we refined it until it was more concrete.

A project timeline is created at the initial stage of the project, and it is updated to ensure that all activities are on track. This timeline helped to evaluate project progress and see the influence of one action on others. We used a Gantt chart to plan and break down the project to get a better overview. In the Gantt chart, we planned the activities that needed to be achieved to deliver a successful project. Each milestone is reflected in the Gantt chart with more detail, and a deadline is assigned to each step. It was mainly used for tracking the progress of the project. Figure 9.2 shows a summary of this project's tasks.

A Graph Database Design for Multi-Domain Model Management			
Tasks Name	**Start Date**	**End Date**	**Duration (Working Days)**
1 Project Initiation	**27-Jun-22**	**20-Jul-22**	**18**
1.1 Project Kickoff	27-Jun-22	27-Jun-22	1
1.2 Setup PSG and regular meeting Schedule	27-Jun-22	28-Jun-22	2
1.3 Environment Setup	28-Jun-22	29-Jun-22	2
1.4 Problem Analysis	28-Jun-22	12-Jul-22	11
1.5 Project Scope Definition	13-Jul-22	20-Jul-22	6
2 Analysis	**21-Jul-22**	**16-Sep-22**	**42**
2.1 Stakeholder Analysis	21-Jul-22	21-Jul-22	1
2.2 Initial Domain Analysis	22-Jul-22	29-Jul-22	6
2.3 Study Related Work	1-Aug-22	19-Aug-22	15
2.4 List of Use Cases	22-Aug-22	31-Aug-22	8
2.5 Requirement Analysis for the project	1-Sep-22	16-Sep-22	12
3 Design and Implementation	**19-Sep-22**	**6-Feb-23**	**101**
3.1 Develop Parser for Simulink Model	19-Sep-22	14-Oct-22	20
3.2 Develop Parser for IBM Rhapsody Model	17-Oct-22	17-Nov-22	24
3.3 Design a Common Meta Ontology (Graph Data Model)	18-Nov-22	25-Nov-22	6
3.4 Develop Frontend/User Interface (UI)	28-Nov-22	13-Dec-22	12
3.5 Develop Parser for Open Modelica	9-Jan-23	20-Jan-23	10
3.6 Further Development and Improvements	31-Jan-23	2/6/2023	5
4 Verification and Validation	**19-Jan-23**	**27-Jan-23**	**7**
4.1 Create test plan for Unit Testing	19-Jan-23	25-Jan-23	5
4.2 Check all the main functionalities	26-Jan-23	27-Jan-23	2
5 Documentation and Presentation	**19-Jan-23**	**3/27/2023**	**48**
5.1 EngD Thesis Report	**19-Jan-23**	**23-Mar-23**	**46**
5.1.1 Create Report Outline	19-Jan-23	20-Jan-23	2
5.1.2 Final Report	23-Jan-23	19-Mar-23	40
5.1.3 Review EngD Report by Supervisor	1-Mar-23	17-Mar-23	13
5.1.4 Review EngD Report by Judith	6-Mar-23	10-Mar-23	5
5.2 Final Presentation	22-Mar-23	28-Mar-23	5
5.3 Project Booklet Article	29-Mar-23	30-Mar-23	2
5.4 Project Ending	31-Mar-23	31-Mar-23	1

Figure 9. 2: Project timeline summary

9.3 *Communication*

A clear and regular communication channel is essential to monitor and manage the project's progress and direction. During the initial meeting, we established the means of communication and frequency with each stakeholder, following their interest and involvement in the project. Each meeting during the project's execution fell into one of the four categories: weekly update meetings, biweekly update meetings, monthly update meetings, and other meetings called on-demand meetings. The communication occurred online through Microsoft Teams, in person at the office, and via email. The regular meeting frequencies and their purposes were as follows:

- Weekly Update Meetings
  - Attendees: Trainee, Project Client
  - Purpose:
    - Demonstrate the overall status and progress of the project
    - Update about completed tasks in the previous week and plans for the next week
    - Identify any challenges and ask project-related questions
    - Identify any misunderstanding as early as possible
    - Ensure that the project is on track
- Biweekly Update Meetings
  - Attendees: Trainee, TU/e Supervisor
  - Purpose:

    - Demonstrate the overall status and progress of the project
    - Update about project tasks, plan, and challenges
    - Ensure the quality of the project results
- Monthly Update Meetings - Project Steering Group (PSG) Meetings
  - Attendees: Trainee, TU/e supervisor, Project Client
  - Purpose:
    - Inform about the project progress at the end of each month
    - Discuss any identified or possible issues
    - Demonstrate the significant updates implemented since the previous PSG meeting
    - Discuss a high-level plan for the next month
    - Get early feedback about the progress
- On-Demand Meetings
  - Attendees: Trainee, any other
  - Purpose:
    - Understand the project context and domain
    - Ask project-related questions to avoid any unexpected delay
    - Succeed in the project on time

9.4 Risk Management

This section describes the risks that were identified during the project. It was essential to maintain a list of risks from the start of the project and propose mitigation action (to reduce the chance of the risk materializing). Table 9.1 describes the risks identified in this project, their occurrence possibility, potential impact and result, and the mitigation strategies applied to manage each.

Table 9. 1: Probable risks and their mitigation plans

Description	**Probability**	**Impact**	**Effect**	**Mitigation Plan**
Lack of domain knowledge and delay in understanding the project context	Medium	High	- Not able to find the best solution - Not being able to complete the project on time	- Consulting with the domain experts and stakeholders - Defining system boundaries to reduce the learning time of the system
Underestimating the project workload	High	High	- Change in expected deliverables - Delay in project completion and lead to project failure	- Conducting comprehensive research - Prototyping at early stages - Continuous feedback meeting
Choosing an inappropriate architectural strategy that cannot satisfy the requirements	Medium	High	- Change in deliverables - Unexpected delay in the progress	- Conducting research and feasibility study - Keeping stakeholders in an early feedback loop
Unavailable of a main stakeholder due to illness	Low	Medium	- Delay for specific deliverables depending on the role - Not delivering the desired artifacts	- Scheduling meetings as early as possible - Providing regular progress reports

				- Trying to find out the alternative in any emergency
Illness for an extended period	Low	High	- Not possible to complete the full scope of the project	- Keep a buffer time between each task - Negotiate requirements with stakeholders according to priority
Miscommunication among stakeholders due to remote work	Low	Medium	- Create a doubtful situation about the project's progress	- Scheduling regular meetings - Working from the office as much as possible
High Priority requirements cannot be satisfied	Medium	High	- Create outcome as an incomplete solution	- Discussing with the stakeholder for a possible solution - Raising the issue as early as possible

10. Conclusion

In this chapter, we wrap up the project and summarize its achievements. It covers the recommendations of possible open directions for future works. Finally, this chapter concludes the report with a reflection on the project from the author's perspective.

10.1 Results and Deliverables

This section describes the result achieved based on the requirements listed in Chapter 5. The project's primary goal is to develop a tool that will include designing a graph database for storing model information of different tools and visualizing several insights. To meet the project's purpose, we developed this tool for the engineers to get an overall and clear idea about the models in a large and complex task in model-based design.

In the following list, the main achievements of this project are mentioned:

- Investigated the models and detected the essential element properties.
  - This tool requires details and relevant information about the model elements and their properties, connectivity with other aspects of the same model, or different model elements. Therefore, we invested most of the development time investigating the models and identifying the essential element properties.
- Developed a parser for extracting model information and storing these data in a file system.
- Designed and developed a graph database for storing data from various models.
- Designed a combined graph data model to represent all model elements
  - We designed a generic graph data model based on each modeling tool's graph data model. We can get a general overview of the graph database by looking at this generic graph data model.
- Identified and implemented several use cases to visualize insights among the models.
  - We chose several use cases – mismatching interfaces, input/output interfaces, change propagation, and hotspot detection based on the available data. We implemented these into our data visualization part. It shows the engineer how the models are internally connected and dependent on each other.
- Stored all source code artifacts, presentation slides, and project reports in the Git repository to deliver the project.

10.2 Recommendations and Future Work

During the project's development, we identified future possibilities, improvements, and features that were not implemented due to the time limitation. The following list can be considered as the improvement points:

- In this tool, we did not implement the model evolution feature yet. The Neo4j labeled property graph model and Cypher query language do not support intrinsic versioning. To version a graph, we must ensure that our application graph data model and queries are version aware. To track changes, we can use time-based versioning [29].
  There are two principles behind time-based versioning. These are:
  - Separate the object from the state that is being linked by a relationship
  - Identify the date and time when the relationship between these two entities changed.
- Integrating more modeling tools to handle more complex project scenarios.
- Store all models in the Git repository and parse models from the repository.

- Adding extensibility to this tool would be a good idea. It can be of two types: reusable code level and model integration level. With this feature, we will be able to add new modeling tools with the minimal development effort.

10.3 Self-Reflection

My last nine-month journey with this project was challenging but also a valuable and pleasing experience. It was an excellent opportunity to combine what I gained during my previous work experience and the knowledge I acquired in the first year of the EngD program. With this project, I experienced a new domain and several new challenges. These challenges provided numerous opportunities to improve my personal and professional skills.

The first challenge was to understand the context and domain of the problem. I spent the first two months on domain study, how models are developed using different modeling tools, and how these models are inter-dependent and inter-connected in heterogeneous cross-domain. I also created a simple proof-of-concept to get better ideas about models. Thus, I gathered experience and knowledge in the relevant domain quickly.

The selection of specific elements and their essential properties concerning the perspective of our project was another challenging task. In the beginning, it was unclear how to achieve this goal. I studied several kinds of literature to know the desired and required elements. It was more critical for me to parse the models from different tools and extract data from them. I developed a tool-specific data parser using several programming languages. For example, I created a parser using MATLAB API and MAT-script for the Simulink model, Java API for IBM Rhapsody SysML models, etc.

Determining the project's requirements and use cases was another significant task. I started creating and analyzing the models of different tools to assess the available data and develop the requirements. In the beginning, the requirements were not more defined. I refined more concrete requirements gradually with the suggestions and feedback from my supervisor, project client, and domain experts.

Furthermore, I gained valuable experience in project management throughout this project. In addition to technical challenges, I faced organizational challenges. I was the leading and only designer and project manager of the entire project. I defined the project roadmap and strategies to tackle difficulties and risks within the project. At the end of the project, my mother had become sick with a serious illness. It was a tough situation for me in this project to keep me motivated. Thus, I gained experience in organizational skills such as project planning, managing risk, and taking initiative and ownership.

Throughout the execution of the project, I developed my technical skills in design and analysis, Python and Java development, and testing. I refactored my design and code several times to increase its quality and make it more extensible. Identifying and prioritizing all requirements and developing the tools based on these were challenging. Additionally, gathering knowledge about the domain and keeping progress with the timeline was highly important. Discussions with stakeholders, learning about their concerns, and regular update meetings helped me overcome all the challenges.

Overall, this project allowed me to broaden my horizons, challenge myself, and improve my skills. I have gained valuable experience and confidence in managing a software project.

Glossary

Abbreviations	**Explanations**
EngD	Engineering Doctorate
GDB	Graph Database
Interface	The input/output of a model element
JSON	JavaScript Object Notation
Model element	Each model consists of several blocks or units. We call each block or unit a model element.
PSG	Project Steering Group
Rhapsody	IBM Rational Rhapsody
ST	Software Technology
SysML	Systems Modeling Language
TU/e	Eindhoven University of Technology

Bibliography

References

[1] Hay, James. "Graph-based approach to managing model relationships."

[2] Fu, Chao, Jihong Liu, and Shude Wang. "Building SysML Model Graph to Support the System Model Reuse." IEEE Access 9 (2021): 132374-132389.

[3] Ramos, Ana Luísa, José Vasconcelos Ferreira, and Jaume Barceló. "Model-based systems engineering: An emerging approach for modern systems." IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42.1 (2011): 101-111.

[4] Dávid, István, et al. "Modeling and enactment support for managing inconsistencies in heterogeneous systems engineering processes." Proceedings of MODELS 2017 Satellite Event, September 17, 2017, Austin, Texas, USA/Burgueño, Loli [edit.]. 2017.

[5] Ö. Babur. “Model Analytics and Management.” English. Proefschrift. Ph.D. thesis. Department of Mathematics and Computer Science, Feb. 2019, pages 1–175. ISBN: 978-90-386-4707-4

[6] Silva Torres, Weslley, M. G. J. van den Brand, and A. Serebrenik. "Model management tools for models of different domains: a systematic literature review." Institute of Electrical and Electronics Engineers, 2019.

[7] Herzig, Sebastian JI, and Christiaan JJ Paredis. "A conceptual basis for inconsistency management in model-based systems engineering." Procedia Cirp 21 (2014): 52-57.

[8] NASA. Report on Project Management in Nasa: Phase Ii of the Mars Climate Orbiter Mishap Report, February 2000. Technical report. Mars Climate Orbiter, Mishap Investigation Board, Mar. 2000

[9] A. Qamar. “Model and Dependency Management in Mechatronic Design.” Ph.D. thesis. KTH Royal Institute of Technology, 2013

[10] R. Hebig, H. Giese, F. Stallmann, and A. Seibel. “On the Complex Nature of MDE Evolution.” In: International Conference on Model Driven Engineering Languages and Systems. Springer. 2013, pages 436– 453

[11] International Council on Systems Engineering (2007). Systems engineering vision 2020.

[12] Schummer, Florian, and Maximilian Hyba. "An Approach for System Analysis with MBSE and Graph Data Engineering." arXiv preprint arXiv:2201.06363 (2022).

[13] McDermott, Thomas A., et al. "Benchmarking the benefits and current maturity of model-based systems engineering across the enterprise." Systems Engineering Research Center (SERC) (2020).

[14] Barbieri, Giacomo, Cesare Fantuzzi, and Roberto Borsari. "A model-based design methodology for the development of mechatronic systems." Mechatronics 24.7 (2014): 833-843.

[15] Minopoli, Stefano, and Goran Frehse. "SL2SX translator: from Simulink to SpaceEx models." Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control. 2016.

[16] Object Management Group. OMG Systems Modeling LanguageTM, 2017. https://www.omg.org/spec/SysML/1.5/PDF. 15, 16

[17] L. Delligatti. SysML Distilled, A Brief Guide to the Systems Modeling Language. Addison-Wesley Professional, 2013. ISBN 9780321927866.

[18] Open-Source Modelica Consortium. “OpenModelica Home Page”(2017). url: https://openmodelica.org (visited on 11/28/2017).

[19] Mengist, Alachew. Methods and Tools for Efficient Model-Based Development of Cyber-Physical Systems with Emphasis on Model and Tool Integration. Vol. 1848. Linköping University Electronic Press, 2019.

[20] Robinson, I. and Webber, J. and Eifrem, E. "Graph Databases." 2nd Edition. June 2015. ISBN: 9781491930892.

[21] Neo4j Database. https://neo4j.com/

[22] Harrison, Guy. Next Generation Databases: NoSQL and Big Data. Apress, 2015.

[23] Db-engine. https://db-engines.com/en/ranking/graph+dbms

[24] Graph temporal versioning. https://medium.com/neo4j/keeping-track-of-graph-changes-using-temporal-versioning-3b0f854536fa

[25] Jeremy Dick, Elizabeth Hull, and Ken Jackson. Requirements engineering. Springer, 2017.

[26] Agile Business Consortium, 2022. [Online]. Available: https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation.

[27] Anne Geraci, Freny Katki, Louise McMonegal, Bennett Meyer, John Lane, Paul Wilson, Jane Radatz, Mary Yee, Hugh Porteous, and Fredrick Springsteel. IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries. IEEE Press, 1991.

[28] P. Hamill, Unit test frameworks: tools for high-quality software development., O'Reilly Media, Inc., 2004

[29] Time-based versioning. https://medium.com/neo4j/keeping-track-of-graph-changes-using-temporal-versioning-3b0f854536fa

[30] Neo4j visualization tools. https://neo4j.com/developer-blog/15-tools-for-visualizing-your-neo4j-graph-database/

[31] van der Heijden, J. J. M. J. "Defining a conversion layer between SysML models and Unity."

[32] Hassan, Ahmed E., and Richard C. Holt. "Predicting change propagation in software systems." 20th IEEE International Conference on Software Maintenance, 2004. Proceedings. IEEE, 2004.

[33] Jongeling, Robbert, et al. "Continuous integration support in modeling tools." 2018 MODELS Workshops: ModComp, MRT, OCL, FlexMDE, EXE, COMMitMDE, MDETools, GEMOC, MORSE, MDE4IoT, MDEbug, MoDeVVa, ME, MULTI, HuFaMo, AMMoRe, PAINS, MODELS-WS 2018, 14 October 2018 through 19 October 2018. Vol. 2245. CEUR-WS, 2018.

[34] P. B. Kruchten, "The 4+ 1 view model of architecture.," IEEE Software 12.6, 1995.

Appendix A. Neo4j Existing Visualization Tools

This section describes some of the existing Neo4j visualization tools. It was essential for us to be aware of the categories of Neo4j's existing tools. Developers built each visualization toolkit with a specific purpose in mind, so we'll have to make sure of the tool's purpose. It was required for us to know which tool matched our need for this project. We found that all graph visualization tools are grouped into four main categories [30]:

1. **Development tools:** Help developers to work with graphs.
2. **Exploration tools:** Help analysts explore data relationships.
3. **Analysis tools:** To reveal trends & discrepancies.
4. **Reporting tools:** To create and organize data reports.

Figure A.1 shows some of the most popular Neo4j graph visualization tools by their primary category. On the vertical axis, it plotted the product type (a Neo4j product, community project, or enterprise software).

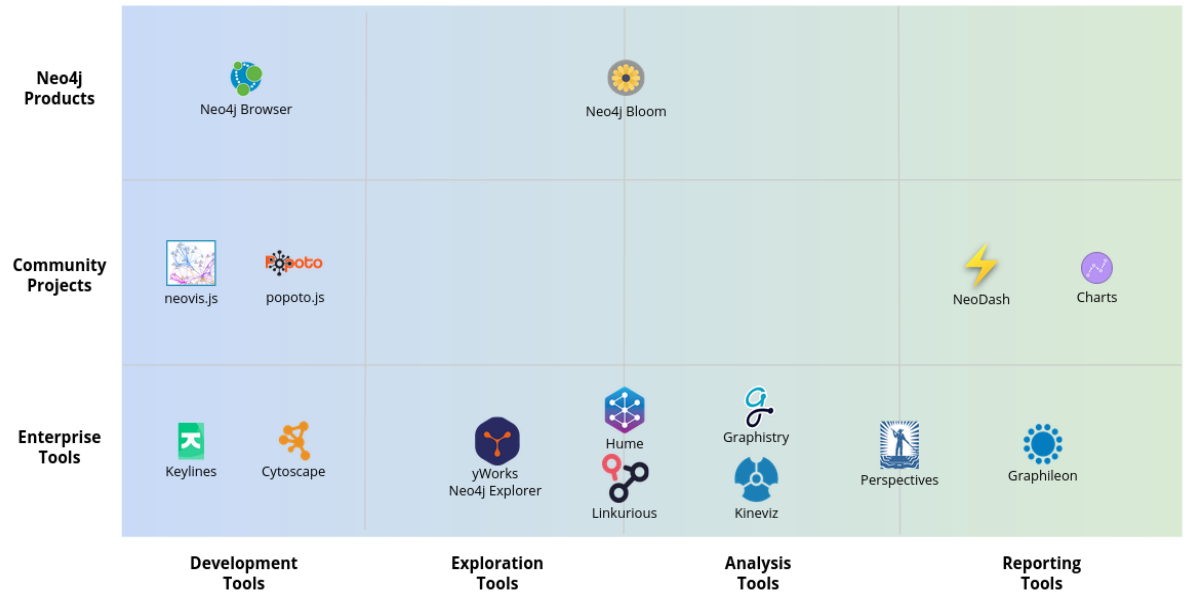

Figure A. 1: Neo4j existing visualization tools [30]

Appendix B. Extracted Data in JSON

The extracted model information is stored in a JSON formatted file after parsing the model file. Figure B.1 shows a sample JSON data extracted from a Simulink model.


```
{"id":"sampleModel","type":"node","labels":["Model"],"properties":{"name":"sampleModel","type":"block_diagram",
"extension":".slx","createdDate":"Tue Oct 05 11:49:12 2021","creator":"ECHOLODO","modifiedBy":"20204920","modifiedDate"
:"Mon Mar 06 22:31:57 2023","modifiedComment":"","description":"","startTime":"0.0","stopTime":"10.0"}}
{"id":"sampleModel/Scope","type":"node","labels":["Block"],"properties":{"name":"Scope","type":"Scope",
"typeDescription":"","description":"Displays input signals with respect to simulation time","numberOfInputPort":1,
"numberOfOutputPort":0,"isCommented":"off"}}
{"id":"","type":"relationship","label":"CONTAINS","properties":{"type":"element"},"start":{"id":"sampleModel"},"end":{
"id":"sampleModel/Scope"}}
{"id":"sampleModel/Scope1","type":"node","labels":["Block"],"properties":{"name":"Scope1","type":"Scope",
"typeDescription":"","description":"Displays input signals with respect to simulation time","numberOfInputPort":1,
"numberOfOutputPort":0,"isCommented":"off"}}
{"id":"","type":"relationship","label":"CONTAINS","properties":{"type":"element"},"start":{"id":"sampleModel"},"end":{
"id":"sampleModel/Scope1"}}
{"id":"sampleModel/Scope2","type":"node","labels":["Block"],"properties":{"name":"Scope2","type":"Scope",
"typeDescription":"","description":"Displays input signals with respect to simulation time","numberOfInputPort":1,
"numberOfOutputPort":0,"isCommented":"off"}}
{"id":"","type":"relationship","label":"CONTAINS","properties":{"type":"element"},"start":{"id":"sampleModel"},"end":{
"id":"sampleModel/Scope2"}}
{"id":"sampleModel/Step","type":"node","labels":["Block"],"properties":{"name":"Step","type":"Step","typeDescription":
"","description":"Output a step.","numberOfInputPort":0,"numberOfOutputPort":1,"isCommented":"off"}}
{"id":"","type":"relationship","label":"CONTAINS","properties":{"type":"element"},"start":{"id":"sampleModel"},"end":{
"id":"sampleModel/Step"}}
{"id":"sampleModel/Step1","type":"node","labels":["Block"],"properties":{"name":"Step1","type":"Step","typeDescription"
:"","description":"Output a step.","numberOfInputPort":0,"numberOfOutputPort":1,"isCommented":"off"}}
{"id":"","type":"relationship","label":"CONTAINS","properties":{"type":"element"},"start":{"id":"sampleModel"},"end":{
"id":"sampleModel/Step1"}}
{"id":"sampleModel/Step2","type":"node","labels":["Block"],"properties":{"name":"Step2","type":"Step","typeDescription"
:"","description":"Output a step.","numberOfInputPort":0,"numberOfOutputPort":1,"isCommented":"off"}}
```


Figure B. 1: JSON data file format of a Simulink model

Below is another representation of the first five data elements of the JSON data shown in Figure B.1:


```
{
"id":"sampleModel",
"type":"node",
"labels":["Model"],
"properties":{
        "name":"sampleModel",
        "type":"block_diagram",
        "extension":".slx",
        "createdDate":"Tue Oct 05 11:49:12 2021",
        "creator":"ECHOLODO",
        "modifiedBy":"20204920",
        "modifiedDate":"Mon Mar 06 22:31:57 2023",
        "modifiedComment":"",
        "description":"",
        "startTime":"0.0",
        "stopTime":"10.0"
        }
}
{
"id":"sampleModel/Scope",
"type":"node",
"labels":["Block"],
"properties":{
        "name":"Scope",
        "type":"Scope",
        "typeDescription":"",
        "description":"Displays input signals with respect to simulation time",
        "numberOfInputPort":1,
        "numberOfOutputPort":0,
```

```
            "isCommented":"off"
            }
}
{
"id":"",
"type":"relationship",
"label":"CONTAINS",
"properties":{
            "type":"element"
            },
"start":{
            "id":"sampleModel"},
"end":{
            "id":"sampleModel/Scope"
            }
}
{
"id":"sampleModel/Scope1",
"type":"node",
"labels":["Block"],
"properties":{
            "name":"Scope1",
            "type":"Scope",
            "typeDescription":"",
            "description":"Displays input signals with respect to simulation time",
            "numberOfInputPort":1,
            "numberOfOutputPort":0,
            "isCommented":"off"
            }
}
{
"id":"",
"type":"relationship",
"label":"CONTAINS",
"properties":{
            "type":"element"
            },
"start":{
            "id":"sampleModel"
            },
"end":{
            "id":"sampleModel/Scope1"
            }
}
```

About the Author

Mohammad Ibrahim received his Bachelor's degree in Information Technology with a Software Engineering major from the University of Dhaka, Bangladesh, in 2012. He also pursued his Master's degree in Software Engineering from the same university in 2014. After graduation, he started a professional career in the software industry in Bangladesh. He is an enthusiastic and versatile Software Engineer having more than five years of experience in the software industry. Lastly, he worked as a Senior Software Engineer at IQVIA, a multinational software firm, and Fortune 500 listed company. In 2019, he decided to take on a new challenge in his life. Therefore, he started to explore new opportunities outside of his country. Then, he joined as a trainee for an Engineering Doctorate (EngD) in Software Technology at the Eindhoven University of Technology in October 2020. During the last two years, he completed multiple projects at well-known companies, including Onera Health, DAF, and ESA, experiencing different roles such as team leader, configuration manager, and engineer. He is interested in Software Architecture, Software Design, and Data Science.

PO Box 513
5600 MB Einhoven
The Netherlands
tue.nl

EngD SOFTWARE TECHNOLOGY

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY