BPMN to Neo4j

Project Report submitted in partial fulfillment of requirements
For the degree of

Bachelor of Engineering
in
Computer Science and Engineering

by

Kushal Das
Class Roll No. - 002010501071
Soumyadeb Misra
Class Roll No. - 002010501088
Rohit Sadhu
Class Roll No. - 002010501074
Arnab Sur
Class Roll No. - 002010501034

under the supervision of

Dr. Mridul Sankar Barik
Assistant Professor
Department of Computer Science and Engineering
Jadavpur University
Kolkata, West Bengal, India

2024

Certificate of Recommendation

This is to certify that the work in this project entitled "BPMN to Neo4j" has been satisfactorily completed by **Kushal Das** (Registration Number 153160 of 2020-2021, Class Roll No. 002010501071, Examination Roll No. CSE248066); **Rohit Sadhu**, (Registration Number 153163 of 2020-2021, Class Roll No. 002010501074, Examination Roll No. CSE248089); **Soumyadeb Mishra**, (Registration Number 153169 of 2020-2021, Class Roll No. 002010501088, Examination Roll No. CSE248046); **Arnab Sur**, (Registration Number 153133 of 2020-2021, Class Roll No. 002010501034, Examination Roll No. CSE248026)). It is a bonafide piece of work carried out under my supervision and guidance at Jadavpur University, Kolkata-700032 for partial fulfillment of the requirements for the awarding of the Bachelor of Engineering in Computer Science and Engineering degree of the Department of Computer Science and Engineering, Jadavpur University, during the academic year 2023-2024.

Dr. Mridul Sankar Barik,
Assistant Professor,
Department of Computer Science and Engineering,
Jadavpur University.
(Supervisor)

Contents

CertificateOfApproval **I**

1 Introduction **1**
1.1 Overview . . . 1
1.2 BPMN . . . 1
1.2.1 BPMN Notation . . . 1
1.2.2 BPMN Diagrams . . . 7
1.3 Neo4j Graph Database . . . 9
1.3.1 Overview . . . 9
1.3.2 How is it different from SQL? . . . 10
1.4 Background and Motivation . . . 10
1.5 Outline of the Report . . . 11

2 Survey of relevant works **12**

3 Methodology **14**
3.1 Problem Statement . . . 14
3.2 Proposed Technique . . . 15

4 Implementation and Results **17**
4.1 Implementation . . . 17
4.1.1 How to execute the code . . . 17
4.1.2 Flow of the Program . . . 18
4.1.3 Code explanations of few important parts of the codebase . . . 20
4.2 Result . . . 21
4.3 Unit Testing Results . . . 21
4.3.1 Unit Tesing is done on 2 type of data files : . . . 21
4.3.2 Specifications of the device used for unit testing : . . . 21
4.3.3 4 types of tests are performed here : . . . 21
4.4 Experimental Result . . . 22
4.4.1 Cooking Diagram . . . 22
4.5 Discussion . . . 26

5 Conclusion and Future Work **27**

List of Figures

4.1 Unit test results on the small data 21
4.2 Unit test results on the large data 22
4.3 BPMN Diagram for cooking a recipe. 22
4.4 Corresponding Neo4j diagram for cooking a recipe. 23
4.5 Showing node properties for one node 24
4.6 Showing the cooking diagram in our own BPMN editor 24
4.7 Showing the graph for the BPMN diagram 25
4.8 Showing the graph after doing a cypher query 25

List of Tables

1.1 BPMN tasks 2
1.2 BPMN flows 3
1.3 BPMN markers 4
1.4 BPMN Data Object Description 5
1.5 BPMN Event Description 5
1.6 BPMN Event Description 6

Abstract

The conversion of a BPMN diagram to a Neo4j database graph allows for the representation and analysis of business processes in a graph database format. This conversion enables the use of graph-based querying and analysis techniques on BPMN models, providing more flexible and efficient ways to explore and optimize business processes. The conversion of a BPMN diagram to a Neo4j database graph combines the structure and semantics of BPMN with the power and flexibility of graph databases, leading to enhanced process understanding, optimization, and automation. To begin the conversion process, it is essential to first understand the structure of a BPMN diagram and how it can be translated into a graph database format. BPMN diagrams consist of various elements such as activities, gateways, events, and flows, each of which can be effectively mapped to nodes and relationships in a Neo4j graph database. Once the mapping is established, the conversion can be carried out by creating corresponding nodes for BPMN elements and defining relationships between them based on the flow and dependencies within the process. Additionally, attributes and properties of BPMN elements can be incorporated as node and relationship properties in the graph database, capturing essential metadata for analysis and querying. Furthermore, the utilization of Cypher, the query language for Neo4j, enables the execution of complex graph-based queries on the BPMN model represented in the Neo4j database. This facilitates insightful analysis, process optimization, and the identification of dependencies and bottlenecks within the business processes.

Chapter 1

Introduction

1.1 Overview

1.2 BPMN

Business Process Model and Notation (BPMN) is a standard for business process modeling that provides graphical notation for specifying business processes in a Business Process Diagram (BPD), based on traditional flowcharting techniques. The objective of BPMN is to support business process modeling for both technical users and business users, by providing notation that is intuitive to business users, yet able to represent complex process semantics. The BPMN 2.0 specification also pro- vides execution semantics as well as mapping between the graphics of the notation and other execution languages, particularly Business Process Execution Language (BPEL). BPMN is designed to be readily understandable by all business stakeholders. These include the business analysts who create and refine the processes, the technical developers responsible for implementing them, and the business managers who monitor and manage them. Consequently, BPMN serves as a common language, bridging the communication gap that frequently occurs between business process design and implementation.

1.2.1 BPMN Notation

A major goal for the development of BPMN was to create a simple and understand- able notation for creating Business Process models, while providing the semantics and underlying mechanisms to handle the complexity inherent in Business Pro- cesses. The approach taken to handle these two conflicting requirements was to organize the graphical aspects of the notation into specific categories. This provides a small set of notation categories so that the reader of a BPMN diagram can easily recognize the basic types of elements and understand the diagram. The various basic BPMN shapes are shown below

BPMN Task/Activity

BPMN 2.0.2	**Task description**
None	No special task type is indicated.
User Task	A User Task is a typical "workflow" task in which a human performer performs the task with the assistance of a software application and could be scheduled through a task list manager of some sort.
Manual Task	A Manual Task is a task that is expected to be performed without the aid of any business process execution engine or application.
Service Task	A Service Task is a task that uses some sort of service, which could be a web service or an automated application.
Call Activity	A Send Task is a simple task that is designed to send a message to an external participant.
Send	A Call Activity is a type of activity within a process. It provides a link to reusable activities.
Script	A Send Task is a simple task that is designed to send a message to an external participant (relative to the process).
Business Rule	A Business Rule Task provides a mechanism for the process to provide input to a Busi- ness Rules Engine and to get the output of calculations that the business rules engine might provide. The input/output specifica- tion of the task will allow the process to send data to and receive data from the Business Rules Engine.

Table 1.1: BPMN tasks

BPMN Flows

BPMN 2.0.2	**Flow description**
Sequence Flow	A Sequence Flow is represented by a solid line with a solid arrow- head and is used to show the order (the sequence) in which activities will be performed in a process or choreography diagram
Association	An Association is represented by a dotted line, which may have a line arrowhead on one or both ends, and is used to associate text and other artifacts with flow objects.
Data Association	A Data Association is represented by a dotted line with a line arrowhead and is used to associate data (electronic or nonelec- tronic) with flow objects. Data Associations are used to show the inputs and outputs of activities.

Table 1.2: BPMN flows

BPMN Marker

BPMN 2.0.2	Marker description
Loop Marker	A Loop Marker is used to represent an activ- ity that will be executed multiple times until the condition is satisfied. The condition can be validated either at the start or end of the activity.
Parallel Multiple Instance Marker	A Parallel Multi-Instance Marker is used to represent an activity that can be executed as multiple instances performed in parallel. The number of instances will be determined through a condition expression that is evaluated at the start of the activity. All the instances will start in parallel and each instance can have different input parameters. The activity, as a whole, is completed after all the instances are completed. However, another expression, if it becomes true, will stop all instances and complete the activity.
Sequential Multiple Instance Marker	A Sequential Multi-Instance Marker represents an activity that is similar to a Parallel Multi- Instance activity, but its instances will be executed in sequence. The second instance will wait until the first instance is completed and so on.
Sequential Multiple Instance Marker	A Sequential Multi-Instance Marker represents an activity that is similar to a Parallel Multi- Instance activity, but its instances will be executed in sequence. The second instance will wait until the first instance is completed and so on.
Annotation Marker	An Annotation Marker is a mechanism for a modeler to provide additional text information (i.e., notes) for the reader of a BPMN diagram. Annotations can be connected to other objects through an Association (see above).

Table 1.3: BPMN markers

BPMN Data Object

BPMN 2.0.2	Data Object Description
Data Object	A Data Object represents the data that are used as inputs and outputs to the activities of a process. Data Objects can represent singular objects or collections of objects.
Data Input	A Data Input is an external data input for the entire process. It is a kind of input parameter.
Data Output	A Data Output is the data result of the entire process. It is a kind of output parameter.
Data Store	A Data Store is a place where the process can read or write data (e.g., a database or a filing cabinet). It persists beyond the lifetime of the process instance.

Table 1.4: BPMN Data Object Description

BPMN Event

BPMN 2.0.2	Event Description
Link (throw and catch)	A Link Event has no significance related to how the Process is performed, but it facilitates the diagram-creation process. For example, you can use two associated links as an alternative to a long sequence flow. There is a throwing Link Event as the "exit point," and a catching Link Event as the "entrance point," and the two events are marked as a pair.
Error (catch)	A catch Error Event is used to capture errors and to handle them. This event can only be used as the start an Event Sub-Process or as a Boundary Event. These events can catch errors thrown by the throw Error Events or errors thrown by a BPM system or services used by the Process.
Error (throw)	A throw Error Event is used set an error to be handled. This event can only be used as an End Event (i.e., never as an Intermediate Event).
Cancel (catch)	Cancel Events can only be used in the context of the transactions. The catch Cancel Events are used as Boundary Events for the transaction Sub-Process, and will trigger the roll back of the transaction (i.e., the Activities of the Sub-Process).
Cancel (throw)	Cancel Events can only be used in the context of the transactions. The throw Cancel Events are only used within a transaction Sub-Process.

Table 1.5: BPMN Event Description

BPMN 2.0.2	Event Description
Conditional (catch)	Conditional Events are used to determine whether to start (or continue) only if a certain condition is true. Like the Timer Event, the Conditional Event can only exist as a catching event. They can be used at the start of a Process or an Event Sub-Process, in the middle of the flow, or as a Boundary Event.
Compensation (catch)	A Compensation Event is used to handle compensation in the process. The catching Compensation Event be triggered as an Event Sub-Process Start Event, or as a Boundary Event.
Compensation (throw)	A Compensation Event is used to handle compensation in the process. The throwing Compensation Event can be used in the middle or end of a Process path.
Signal (start)	Catching Signal Events are used for receiving signals. They are a generic, simple form of communication and exist within pools (same participant), across pools (different participants), and across diagrams. They can be used at the start of a Process or an Event Sub-Process, in the middle of the flow, or as a Boundary Event.
Signal (end)	Throwing Signal Events are used for sending signals. They are a generic, simple form of communication and exist within pools (same participant), across pools (different participants), and across diagrams. They can be used in the middle or end of a Process path.
Multiple (catch)	The Multiple Event is used to summarize several event types with a single symbol. The event is triggered if any one of those types is satisfied. They can be used at the start of a Process or an Event Sub-Process, in the middle of the flow, or as a Boundary Event.
Multiple (throw)	The Multiple Event is used to summarize several event types with a single symbol. When this is event is reached, then all the event types are thrown. They can be used in the middle or end of a Process path.
Parallel Multiple (catch)	The Parallel Multiple Event is used to summarize several event types with a single symbol. The difference between this event and the Multiple Event is that the Parallel Multiple is only triggered if all of those types are satisfied. They can be used at the start of a Process or an Event Sub-Process, in the middle of the flow, or as a Boundary Event.
Terminate (throw)	The Terminate End Event is the “stop everything” event. When a Terminate End Event is reached, the entire process is stopped, including all parallel activities.

Table 1.6: BPMN Event Description

1.2.2 BPMN Diagrams

Business Process Modeling is used to communicate a wide variety of process con- figurations to a wide variety of audiences. Thus, BPMN was designed to cover many types of modeling and allow the creation of end-to-end Business Processes. The structural elements of BPMN allow the viewer to be able to easily differentiate between sections of a BPMN Diagram. There are three basic types of submodels within a BPMN modeling environment:

- Processes (Orchestration), including:
  - Private non-executable (internal) Business Processes.
  - Private executable (internal) Business Processes.
  - Public Processes.
- Choreographies.
- Collaborations, which can include Processes and/or Choreographies.
  - A view of Conversations.

Private Business Processes

Private business processes refer to the operations and procedures that occur within a particular organization. These processes are often referred to as workflows or BPM (Business Process Management) processes. Another term commonly used, especially in the realm of web services, is the orchestration of services.There are two main categories of private processes: executable and non-executable.

Public Business Processes

A public process illustrates the interactions between one process or participant and another. It only includes activities and events that serve to communicate with other participants, essentially acting as touchpoints between them. Internal activities within the private business process are not visible in the public process. Therefore, the public process displays the message flows and their sequence necessary for interacting with that process to the external world. Public processes can be modeled independently or within a collaboration to depict the directional flow of messages. It's worth noting that in BPMN 1.2 (2009 release), this type of process was referred to as "abstract."

BPMN Collaboration

A collaboration illustrates the interactions among two or more business entities. Typically, a collaboration includes two or more pools, each representing a participant in the collaboration. The exchange of messages between these participants is depicted by message flows connecting the pools or the objects within them. These messages associated with the message flows can also be visually represented. In a collaboration, the interaction can be shown as two or more public and/or private processes communicating with each other. Alternatively, a pool within the collaboration may be empty, essentially acting as a "black box." Choreography elements may be displayed between the pools, intersecting the message flows between them. In essence, collaborations can involve various combinations of pools, processes, and choreography elements, allowing for flexibility in representing complex interactions among business entities.

BPMN Choreography

A self-contained choreography, devoid of pools or orchestration, serves as a blueprint outlining the expected behavior between interacting participants. Essentially, it functions as a procedural contract delineating how different participants should interact with each other. Unlike a standard process that operates within a pool, a choreography is situated between pools or participants. Visually, it resembles a private business process, comprising a network of activities, events, and gateways. However, its distinctiveness lies in its nature: the activities represent interactions involving two or more participants and signify a set of message exchanges. Unlike a typical process, a choreography lacks a central controller, responsible entity, or observer overseeing the process. Instead, it embodies a decentralized approach, where each participant plays a role in the interaction without a single entity orchestrating the sequence of events.

BPMN Conversation

The Conversation diagram serves as a specialized and informal representation of a Collaboration diagram. However, unlike a traditional collaboration diagram, the pools in a Conversation diagram typically do not contain processes, and choreography elements are usually not placed between the pools. Instead, each individual conversation within the diagram represents a logical relationship of message exchanges. These logical relationships often revolve around specific business objects or entities of interest, such as "Order," "Shipment and Delivery," or "Invoice." Essentially, the Conversation diagram provides a high-level modeling overview depicting a set of interconnected conversations that reflect distinct business scenarios. For instance, in the realm of logistics, scenarios like stock replenishment entail various conversations such as the creation of sales orders, carrier assignments for shipments, customs clearance, payment processing, and exception handling. This diagram offers a panoramic view of the different conversations relevant to the domain, providing a comprehensive understanding of the interactions involved.

1.3 Neo4j Graph Database

1.3.1 Overview

Neo4j is a leading graph database management system designed to efficiently store, manage, and query highly interconnected data. Unlike traditional relational databases that use tables to represent data and relationships, Neo4j utilizes a graph-based model, which is particularly well-suited for scenarios where relationships between entities are as important as the entities themselves.

In Neo4j, data is represented as nodes, which can be thought of as entities, and relationships, which define connections between nodes. Each node and relationship can have properties that store additional information. This graph structure allows for the representation of complex relationships in a natural and intuitive way.

The key advantage of Neo4j lies in its ability to handle and traverse relationships with high performance. Queries that involve navigating through relationships are typically more efficient in a graph database than in traditional relational databases. This makes Neo4j an ideal choice for applications that involve social networks, fraud detection, recommendation systems, network analysis, and other scenarios with intricate relationships.

Neo4j provides a query language called Cypher, specifically designed for expressing graph patterns. Cypher allows developers and data analysts to interact with the graph database and retrieve information using a syntax that closely aligns with graph patterns.

The graph database also offers ACID (Atomicity, Consistency, Isolation, Durability) compliance, ensuring data integrity and reliability. Neo4j can be deployed in various environments, including on-premises servers, cloud services, or as part of a larger application architecture.

Therefore, Neo4j is a powerful and versatile graph database that excels at handling interconnected data, making it a valuable tool for applications requiring sophisticated relationship modeling and traversal. Its adoption continues to grow across industries as organizations recognize the importance of leveraging graph structures to unlock valuable insights from their data.

1.3.2 How is it different from SQL?

Neo4j, a graph database, differs from traditional SQL (Structured Query Language) and NoSQL (Not Only SQL) databases in terms of data modeling, query language, and the underlying architecture. NoSQL refers to a broader concept whole together and Neo4j is another type of NoSQL database. It has these salient features:

- **Data Model:** Neo4j is a graph database, and its data model is based on nodes, relationships, and properties. This allows for the representation of complex relationships between entities, making it suitable for scenarios where relationships are as important as the data itself.

  Whereas, SQL databases are relational databases, meaning they use a table-based data model; data is organized into tables with predefined schema, and relationships between tables are established using foreign keys and NoSQL databases encompass a variety of models, including document-oriented, key-value, column-family, and graph databases.

- **Query Language:** Neo4j uses the Cypher query language, specifically designed for querying graph databases. Cypher allows developers to express complex graph patterns and navigate relationships with ease.

  Whereas, SQL databases use the SQL query language, which is primarily designed for working with tabular data like SELECT, INSERT, UPDATE, and DELETE etc. statements and each NoSQL database has its own query language or API.

- **Relationships:** Relationships are fundamental to the data model in Neo4j. Traversing relationships is a core strength of Neo4j, making it efficient for scenarios where relationships are a key aspect of the data. In Neo4j relationships are established using edges between nodes (which are also known as sequence flows).

  Whereas, in SQL relationships are typically established through foreign key constraints between tables and other NoSQL databases may denormalize data to support relationships, while others may require additional programming logic.

1.4 Background and Motivation

- Lack of comprehensive reasoning and analysis tools for BPMN diagrams.
- Proposal to leverage property graph representations with nodes and edges.
- Ability to query and analyze BPMN processes represented as graphs.
- Aim to convert BPMN diagrams into graph structures for enhanced analysis.
- Utilization of graph databases like Neo4j for efficient querying and analysis.
- Motivated by the need to empower users with interactive exploration and understanding of complex business processes.

1.5 Outline of the Report

This is report is organized as follows:

- **Chapter 1 - Introduction**: The project aims to convert BPMN diagrams to Neo4j graphs, bridging business process modeling with graph database technology. The introduction provides an overview of the project's significance and goals, delving into the essential components of BPMN notation and diagrams, along with an introduction to Neo4j and its distinctions from traditional databases like SQL and NoSQL. Motivation behind the project and an outline of the report's structure are also discussed.
- **Chapter 2 - Survey of Relevant Works**: This section entails a comprehensive review of existing literature, research papers, and projects pertaining to BPMN conversion to Neo4j, offering insights into the current state of the field and identifying potential gaps for this project to address.
- **Chapter 3 - Methodology**: Here, the project's methodology is elucidated, delineating the problem statement and presenting the proposed technique for converting BPMN diagrams to Neo4j graphs, including the underlying principles and processes involved.
- **Chapter 4 - Implementation and Results**: Detailed implementation of the proposed technique is outlined, offering insights into program execution flow, code snippets, and key modules. Results obtained from the implementation, along with unit testing outcomes and experimental results from case studies, are presented and analyzed in this section. Challenges faced during implementation and discussions on the obtained results are also included.
- **Chapter 5 - Conclusion and Future Work**: The section summarizes the key findings and conclusions drawn from the project, highlighting its contributions and potential implications. Additionally, suggestions for future work or enhancements to the proposed technique are provided, paving the way for further research and development in this domain.

Chapter 2

Survey of relevant works

1. **A BPMN-based language for modeling corporate communications[1]**

   This paper seeks to introduce the CCML language extension for clear and concise visualization of organizational communication processes with BPMN. It´s objective is to apply common rules and procedures that will allow establish communications and support the interoperability of different interactions and technologies. The add-on contains the blocks like Conversation nodes, Message Flows and Sub-conversations that are reviewed based on public communication models. CCML was modelled with the XML Schema Definition in the structured manner using the meta-models for Conversation Nodes and Participants as an extension to BPMN. It is mentioned in the document that BPMN extension, communication theories, and corporate communication practices are in this regard vital. This signifies that in the organizations communication effectiveness is of paramount importance. It describes the approach to languages, syntax, case, notation, as well as the examples of DFD presented in the report. The paper points out the necessity of cognitive-effective notation with clearly represents symbols for the purposes to make diagrams emotional and comprehensive. Generally, the CCML distinction is specifically created to ensure that corporate communications are representable, technology-autonomic, and adaptable, serving contemporary communication requirements.

2. **Privacy-enhanced BPMN: enabling data privacy analysis in business processes models[2]**

   The paper focuses on PE–BPMN, an extension of BPMN which helps in modelling processes that employ privacy-enhancing technologies. This analyzes and shares the issues related to the protection and requirements of the privacy among the stakeholders. This proposal introduces a stratified model for the PeTX and shows the approaches for the analysis of the information disclosures in enriched models with PE–BPMN.

   Personal data privacy concerns have become magnified, with GDPR coming into force to prop up this aspect. Yet, it is an area that is not being studied by measuring privacy degrees in business processes. This gap is covered by this paper which creates PET–oriented goals and objectives using BPMN taxonomy as a basis. The taxonomy includes goals like secure communication, secure data and preserving privacy, secure entity authentication, and privacy–

aware computing.

The PE–BPMN syntax consists of protection elements such as security, anonymity, data security, entity authentication, privacy-aware computation, and intervention. This set of features facilitates modeling privacy-enhancing activities that exist in business process models.

The paper continues showing PE–BPMN being used in a situation where relevant data is not only collected by mobile phone app but also some information that is entered by the involved organization like a hospital. The situation is modeled on a generic basis, and participants take different roles which represent some specific goals and privacy concerns. The paper also demonstrates creating a model of abstracted stereotypes, for example the concrete PETs just like encryption, secret sharing, or Intel SGX. Such explicit stereotypes make possible the use of various privacy-boosting tools and considerations of trade-offs between various such tools.

On the other hand, the paper explains the visibility matrices, communication matrices, and data dependency matrices tools used for information disclosure analysis techniques at the end. Such methodology is employed for the identification and examination of information, and messaging interaction in a PE–BPMN model.

3. **A Framework for Querying Graph–Based Business Process Models[3]**

A framework for querying and reusing graph–based business process models is presented by Sakr and Awad, utilizing BPMN–Q, a visual query language. BPMN–Q extends BPMN notation for process modeling and facilitates structural queries, matching process model graphs to query graphs. The framework incorporates a semantic query expander for flexibility in retrieving models with high similarity to queries. Relational database management systems (RDBMS) are leveraged, employing a decomposition-based and selectivity-aware query processing mechanism for efficient graph–based queries. This approach addresses the challenge of effectively reusing process models, enhancing business process modeling tasks by providing an intuitive query language and efficient query processing techniques.

Chapter 3

Methodology

3.1 Problem Statement

The primary objective of this project is to develop a seamless integration between BPMN.js diagrams and Neo4j graph databases, creating a robust and efficient workflow for visualizing and managing business processes. The project aims to leverage BPMN.js, a powerful BPMN (Business Process Model and Notation) modeling library, to capture and design business processes graphically. Subsequently, the project seeks to establish a connection with Neo4j, a leading graph database, to store and represent these business processes in a structured and interconnected manner.

Specifically, the project aims to achieve the following objectives:

- **Integration of BPMN.js with Neo4j:**

  Implementing a mechanism to translate BPMN.js diagrams into a format compatible with Neo4j's graph representation and establish a secure and efficient connection between BPMN.js and Neo4j, ensuring seamless data transfer.

- **Data Mapping and Schema Design:**

  Define a mapping strategy to convert BPMN.js elements, such as tasks, gateways, and events, into appropriate nodes and relationships in the Neo4j graph. Design a Neo4j schema that effectively captures the structural and relational aspects of BPMN.js diagrams.

- **Performance Optimization:**

  Conduct performance tuning to ensure efficient data storage, retrieval, and visualization, even with large and complex BPMN.js diagrams.

- **Documentation and User Guide:**

  Prepare comprehensive documentation that outlines the integration process, data mapping strategies, and usage guidelines. Create a user guide to assist stakeholders in effectively utilizing the integrated BPMN.js to Neo4j solution.

- **Testing and Validation:**

  Conduct thorough testing to validate the correctness of data integration, graph representation, and user interface functionality and subsequently address and resolve any issues or inconsistencies identified during the testing phase.

- **Scalability and Extensibility:**

  Design the integration solution with scalability and extensibility in mind, allowing for future enhancements and accommodating the evolving needs of business process management.

  By achieving these objectives, the project aims to provide a comprehensive solution that empowers users to seamlessly transition from BPMN.js modeling to Neo4j graph representation, fostering a more efficient and interconnected approach to managing and analyzing business processes.

3.2 Proposed Technique

Converting a BPMN (Business Process Model and Notation) diagram to Neo4j involves several steps, including extracting the relevant information from the BPMN diagram, representing it in XML, and then transforming that XML data into a graph database model suitable for Neo4j. Here's a breakdown of the process:

- **Extract BPMN Information:**

  BPMN diagrams represent business processes using graphical elements such as tasks, gateways, events, and flows. These elements need to be extracted and analyzed to understand the structure and flow of the process.

- **XML Representation:**

  BPMN diagrams can be serialized into XML format using the BPMN XML Schema. This XML representation captures all the necessary information about the process, including its activities, gateways, sequence flows, and other attributes.

- **Mapping to Graph Model:**

  Once you have the BPMN information in XML format, you need to map it to a graph model suitable for Neo4j. In Neo4j, nodes represent entities, and relationships represent connections between entities. Each element in the BPMN diagram (e.g., tasks, gateways) can be mapped to nodes in the graph, and the connections between them can be represented as relationships.

- **Node and Relationship Mapping:**

  - **Nodes:**
    Each BPMN element (task, gateway, event) can be mapped to a corresponding node in Neo4j. For example, a task in the BPMN diagram can be mapped to a "Task" node in Neo4j.
  - **Relationships:**
    The connections between BPMN elements (sequence flows) can be mapped to relationships in Neo4j. For example, a sequence flow between two tasks can be represented as a relationship between the corresponding task nodes.

- **Property Mapping:**

  Attributes of BPMN elements can be mapped to properties of nodes and relationships in Neo4j. For example, the name of a task can be stored as a property of the corresponding node.

- **Data Import:**

  Once you have mapped the BPMN diagram to a graph model in XML format, you can import this data into Neo4j. Neo4j provides tools and APIs for importing data from various formats, including XML.

- **Querying and Analysis:**

  With the BPMN process represented as a graph in Neo4j, you can now query the data using Cypher (Neo4j's query language) to analyze the process, identify bottlenecks, optimize workflows, and gain insights into process execution.

Chapter 4

Implementation and Results

4.1 Implementation

4.1.1 How to execute the code

Here's how to run our frontend application:

1. First clone the git repository:

```
git clone https://github.com/das-kushal/bpmn-to-neo4j-frontend
```

2. Install the dependencies in your system:

```
yarn
```

OR

```
npm install
```

3. Create the .env.local file (environment file) and write your credentials :

```
VITE_NEO4J_URL="<your-neo4j-url>"
VITE_NEO4J_USERNAME="<your-neo4j-username>"
VITE_NEO4J_PASSWORD="<your-neo4j-password>"
```

4. Now run the project with this command:

```
yarn dev
```

OR

```
npm run dev
```

Here's how to use our package:

1. First install the package in your project:

```
npm install bpmn-to-neo4j
```

2. Then import main function from 'bpmn-to-neo4j' package:

```
import main from 'bpmn-to-neo4j';
```

3. Call the main function by providing the **(XML), neo4jURL, neo4jUsername, neo4jPassword** respectively:

```
main(XML string, '<bolt://localhost:XXXX>', 'neo4j','12345');
```

4. Then run this file (say **index.js**):

```
node index.js
```

5. When in the console *'Query completed and Committed'* are printed then go to neo4j browser to see the neo4j graph:

```
******INITIALISING SESSION**********
Query completed
committed
```

6. Then in neo4j browser type this below cipher query to see the whole graph created corresponding to the bpmn diagram :

```
 match (n) return  n;
```

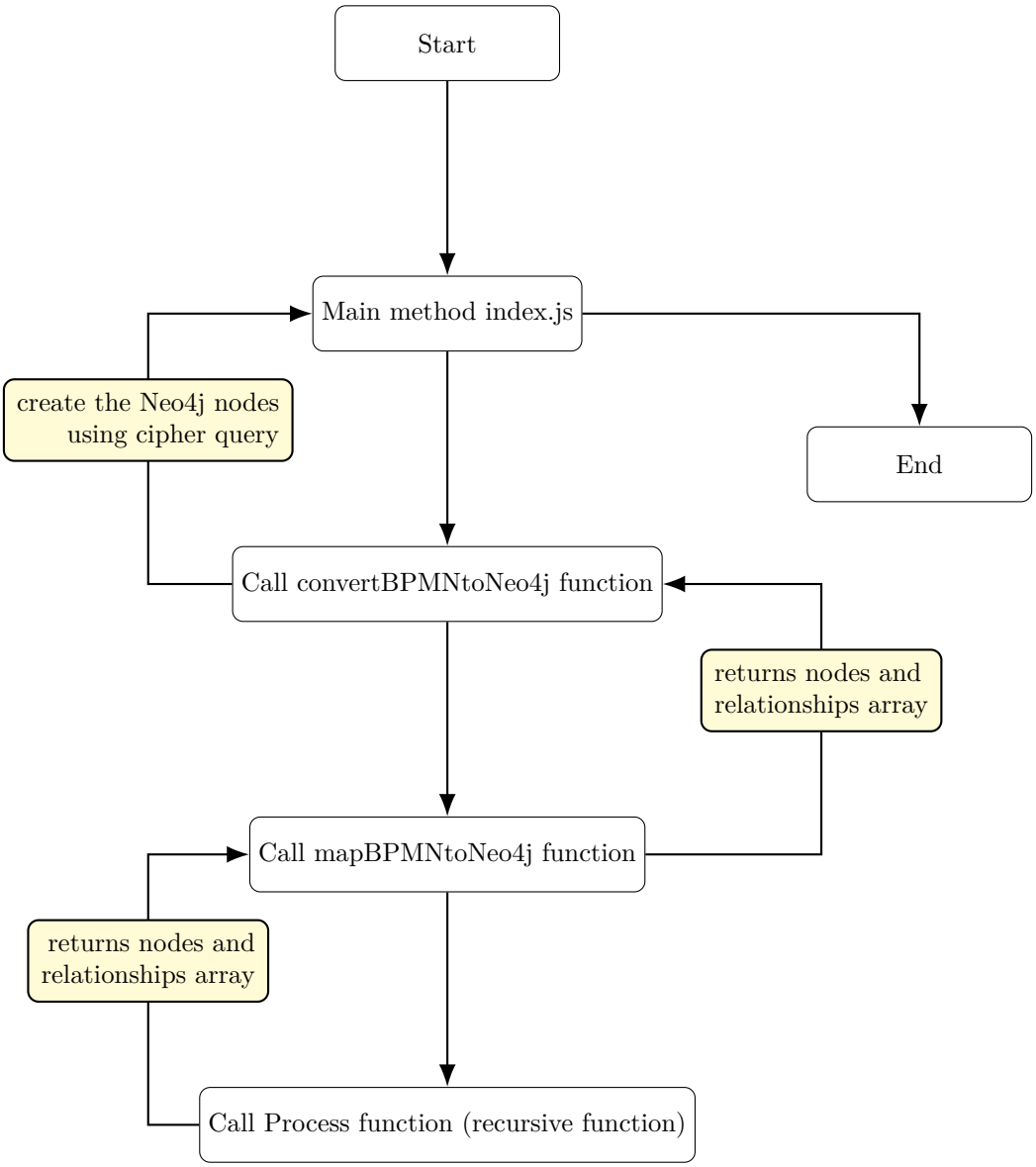
4.1.2 Flow of the Program

1. Our program starts at the *index.js* file of the driver program which calls the main method of the *index.js* file of the source program.
2. Now the main method of *index.js* file creates Neo4j driver and calls the **convertBPMNtoNeo4j** function.In the index file, we have to provide the data and the Neo4j driver is created using the createNeo4jDriver function where we need to give the Neo4j username and password and the URL.
3. Then the convertBPMNtoNeo4j calls **mapBPMNtoNeo4j** function to build the array of nodes and relationships of Neo4j.
4. In the map file we first check for some edge cases like if the data is null or undefined or maybe it is empty and then we initialise them to empty arrays so that they can contain the nodes and the relationships .
5. If we look in the XML file, we see that there are some references and there are sequence flows and there are incoming & outgoing nodes.
6. So we check for the type of the nodes and take necessary action like inserting them into the corresponding arrays.
7. Different arrays are created for different purposes.

8. We have sequence flow array to store the sequence flows and we stored the sourceRef and targetRef in the associations array. Finally for each task we push it in neo4jData.nodes array and we store the relationships in neo4jData.relationships array.

9. And then we return them to the **convertBPMNtoNeo4j** file to write the cipher query to create those nodes and relationships.

Basic flowchart of the flow of the BPMN to Neo4j package :



4.1.3 Code explanations of few important parts of the codebase

1. (a) Here we read the BPMN (XML) file and then we create the neo4jDriver using the function we create in the **DBConnect.js** file and we call the *convertBPMNtoNeo4j* function.

2. (a) Here in this code snippet we have used the *neo4j.driver* function to make a neo4jdriver and we return it from this function

3. (a) Here in this code snippet, we define the neo4jData to contain the nodes and relationships
   (b) In *bpmnData.rootElement[0]* we have the process object and references object is in *bpmnData.references*
   (c) Then we call the Process function which is a recursive function *(to handle subprocess)* to populate the neo4jData object.

4. (a) Then we initialise the arrays to hold tasks, annotations and associations

5. (a) If artifact is of type textAnnotation then we push it in the textAnnotations array
   (b) And if the reference element is of type Association then push it in associations array
   (c) Here the task node has an activity id and text annotation has annotation id so ***association = {activity (SourceRef), annotation (TargetRef)}***

6. (a) Here in this code sample first part we are mapping the dataAssociation id to the database reference id.
   (b) Next part shows a code snippet for storing Sequence Flows in the sequenceFlows array and also shows how we are extracting the type of the BPMN element,converting the data into a node for Neo4j and sorting in tasks array.
   (c) In the next snippet we map the annotations to tasks based on associationss

7. (a) This snippet shows how a relationship is created between the datastore reference i.e the actual database and the task to which the database is connected or the dataobject reference i.e the reference to the actual data and how the mapping is done betweeen the datastore reference or dataobject reference and the data association.

8. (a) Here we check if the node has attributes like annotations , if present we append the annotations to the node properties
   (b) Then we return the cipher query to create a Neo4j node
   (c) We also return the cipher query to create the relationships between the nodes
   (d) Finally we initialise a Neo4j session and begin the transaction , run each query .
   (e) If the transaction is committed then we return successful else we roll back completely as transaction is ***atomic***, at the end we close the session

4.2 Result

4.3 Unit Testing Results

4.3.1 Unit Tesing is done on 2 type of data files :

- Small Data set (having about 1000 lines of data ~30KB)
- Relatively large Data set (having about 5000 lines of data ~145KB)

4.3.2 Specifications of the device used for unit testing :

M1 Macbook Air 2020 , 16GB RAM and 512GB SSD , 8 core CPU(4 performance and 4 efficiency)

4.3.3 4 types of tests are performed here :

- Asserting that the BPMN data is properly converting in Neo4j data
- Asserting that our code does not break even if BPMN data is empty
- Asserting that our code does not break even if BPMN data is **invalid(undefined or NULL)**
- Our program should work within reasonable time frame *(here taken threshold to be 1sec)*

```
PASS ./MapBPMNtoNeo4j.test.js
  MapBPMNtoNeo4j Function
    ✓ should properly convert BPMN data to Neo4j data (17 ms)
    ✓ should handle empty BPMN data
    ✓ should handle invalid BPMN data (1 ms)
    ✓ should execute within a reasonable time frame with large input (2 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.278 s, estimated 1 s
Ran all test suites matching /MapBPMNtoNeo4j.test.js/i.
```

Figure 4.1: Unit test results on the small data

Here we can see on the small data the unit tests takes about ***0.278sec*** which is pretty fast

```
PASS ./MapBPMNtoNeo4j.test.js
  MapBPMNtoNeo4j Function
    ✓ should properly convert BPMN data to Neo4j data (17 ms)
    ✓ should handle empty BPMN data (1 ms)
    ✓ should handle invalid BPMN data
    ✓ should execute within a reasonable time frame with large input (5 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.39 s, estimated 1 s
Ran all test suites matching /MapBPMNtoNeo4j.test.js/i.
```

Figure 4.2: Unit test results on the large data

Here we can see on the large data the unit tests takes about ***0.39sec*** which is pretty fast but takes more time than the previous smaller data

4.4 Experimental Result

4.4.1 Cooking Diagram

*Showing how **annotations** can be incorporated:*

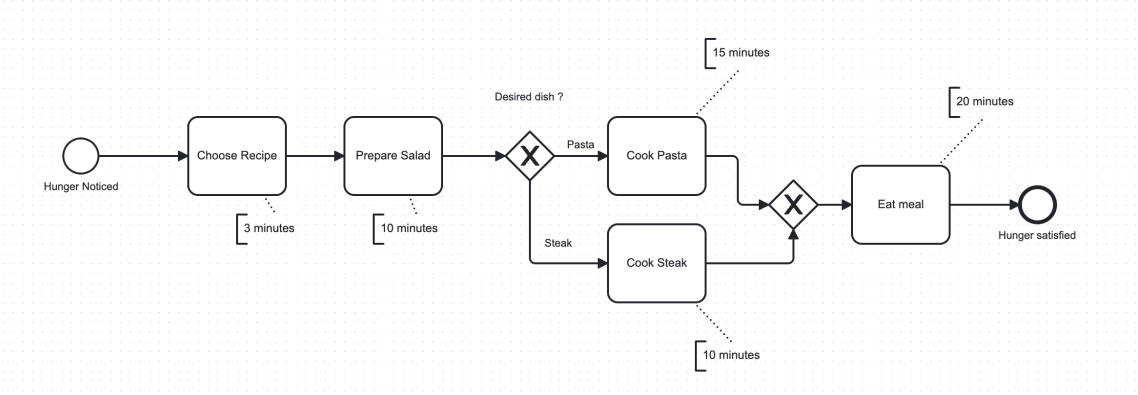


Figure 4.3: BPMN Diagram for cooking a recipe.

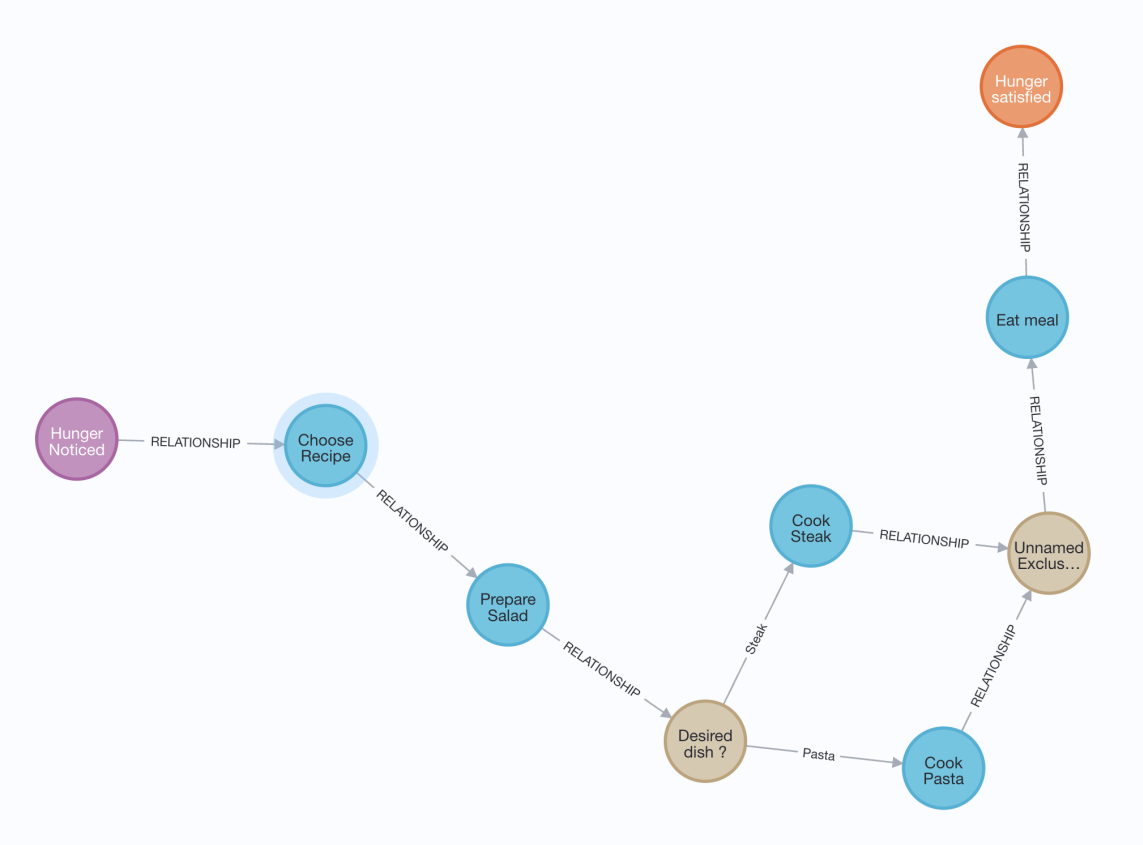


Figure 4.4: Corresponding Neo4j diagram for cooking a recipe.

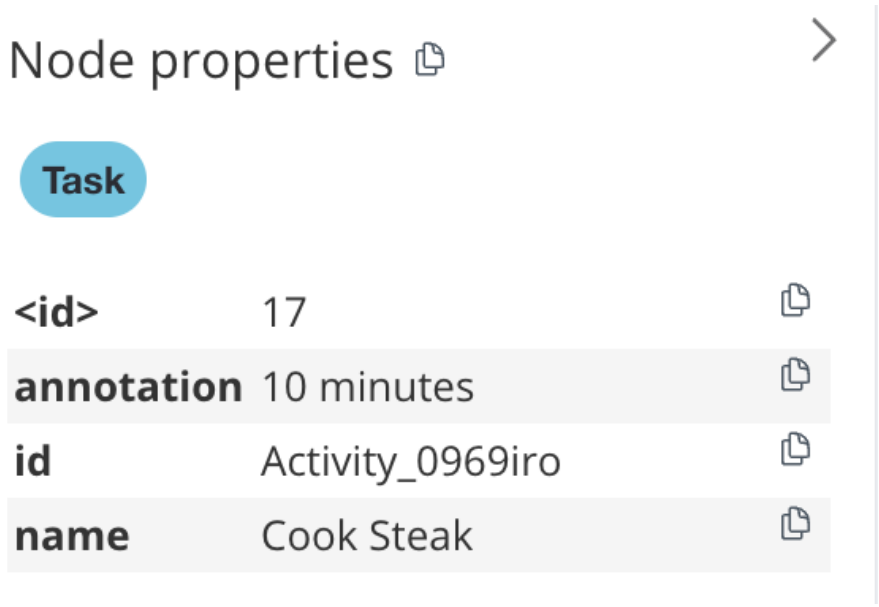


Figure 4.5: Showing node properties for one node

Taking it to frontend

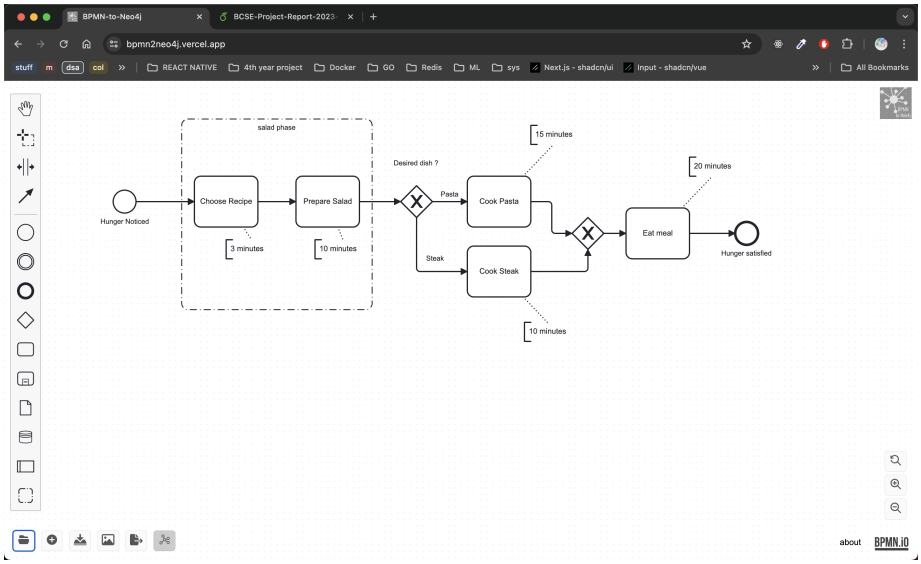


Figure 4.6: Showing the cooking diagram in our own BPMN editor

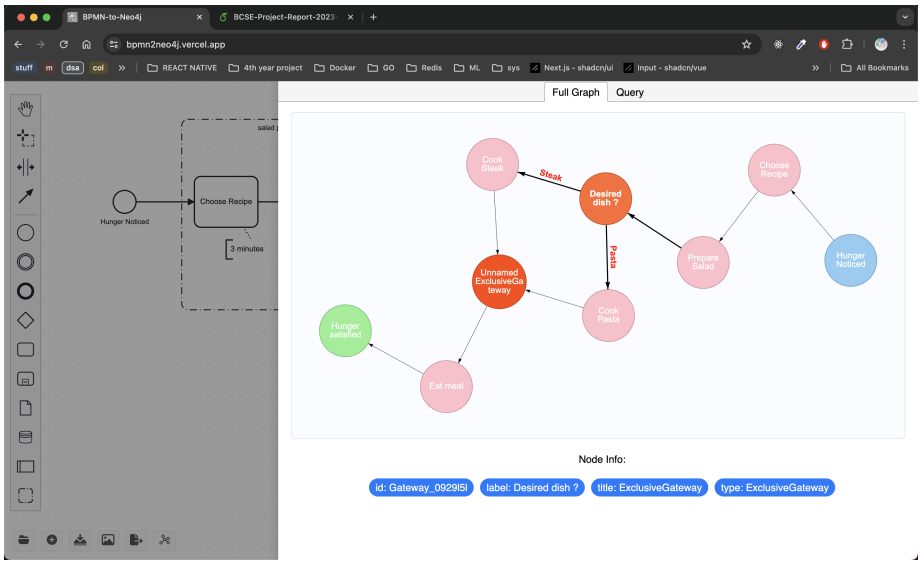


Figure 4.7: Showing the graph for the BPMN diagram

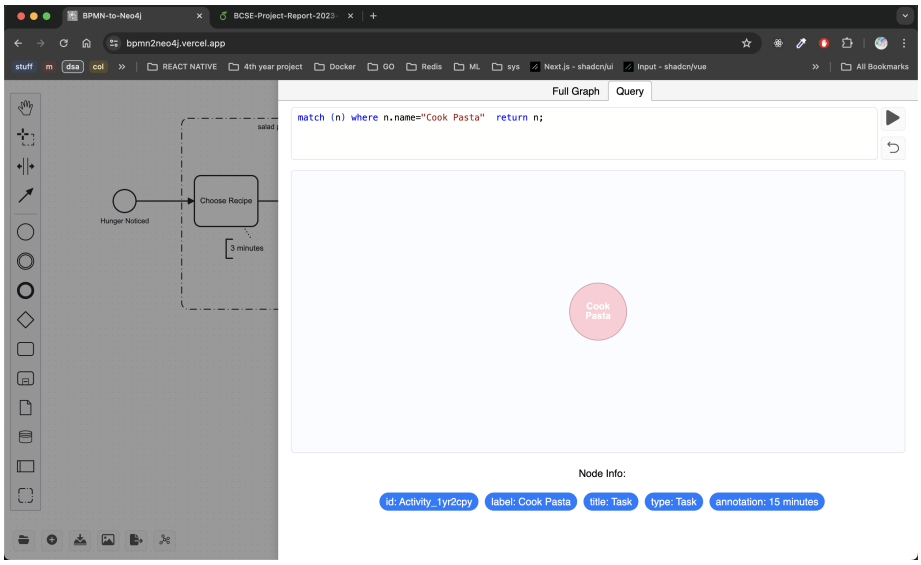


Figure 4.8: Showing the graph after doing a cypher query

```
<?xml version="1.0" ...>
  <bpmn2:process id="Process_1">
    <bpmn2:startEvent id="Event_0d5lr0o" name="Hunger Noticed">
      <bpmn2:outgoing>Flow_0c3sy3l</bpmn2:outgoing>
    </bpmn2:startEvent>
```

```
    <bpmn2:task id="Activity_1p8iqcn" name="Choose Recipe">
      <bpmn2:incoming>Flow_0c3sy3l</bpmn2:incoming>
      <bpmn2:outgoing>Flow_0k3nefy</bpmn2:outgoing>
    </bpmn2:task>
    <!-- HIGHLIGHTED PART -->
      <bpmn2:textAnnotation id="TextAnnotation_1q695hs">
      <bpmn2:text>3 minutes</bpmn2:text>
    </bpmn2:textAnnotation>
    <bpmn2:association id="Association_041a4y2" sourceRef="Activity_1p8iqcn"
        targetRef="TextAnnotation_1q695hs" />

    <bpmn2:sequenceFlow id="Flow_0c3sy3l" sourceRef="Event_0d5lr0o"
        targetRef="Activity_1p8iqcn" />
    <bpmn2:task id="Activity_1xy0xyl" name="Prepare Salad">
      <bpmn2:incoming>Flow_0k3nefy</bpmn2:incoming>
      <bpmn2:outgoing>Flow_0bvuhfg</bpmn2:outgoing>
    </bpmn2:task>
    ...
    <bpmn2:exclusiveGateway id="Gateway_1oizb5t">
      <bpmn2:incoming>Flow_0awc667</bpmn2:incoming>
      <bpmn2:incoming>Flow_0ikzp16</bpmn2:incoming>
      <bpmn2:outgoing>Flow_055upvu</bpmn2:outgoing>
    </bpmn2:exclusiveGateway>
   ...
    <bpmn2:endEvent id="Event_0vdzu5q" name="Hunger satisfied">
      <bpmn2:incoming>Flow_0flu9vl</bpmn2:incoming>
    </bpmn2:endEvent>
    <bpmn2:sequenceFlow id="Flow_0awc667" sourceRef="Activity_0969iro"
        targetRef="Gateway_1oizb5t" /> ...
  </bpmn2:process>
</bpmn2:definitions>
```

Listing 4.1: Some parts of the XML file of cooking diagram

4.5 Discussion

The project report highlights the conversion of BPMN 2.0 diagrams into graph structures for querying and analysis within databases such as Neo4j. The process entails extracting BPMN data, encoding it in XML, and mapping it to a graph model suitable for Neo4j. Each element within the BPMN diagram is translated into a node, while connections between elements are represented as relationships. This graph representation facilitates the identification of process bottlenecks, workflow optimization, and insights into execution. By leveraging the capabilities of graph databases, the analysis process is streamlined, enabling users to efficiently explore and comprehend complex business processes interactively.

Chapter 5

Conclusion and Future Work

This project successfully unified the intuitive process modeling of BPMN.js with the powerful data representation of Neo4j, creating a versatile solution for business process management. Key achievements include:

- Seamless integration: A robust connection allows translating BPMN diagrams into interconnected Neo4j graphs, capturing the essence of each element accurately.
- Enhanced user interface: A unified platform combines BPMN.js modeling with Neo4j interaction, enabling efficient design, storage, retrieval, and analysis of processes.
- Performance optimization: Scalability and responsiveness are ensured even for complex diagrams, making the solution practical for diverse scenarios.
- Comprehensive documentation: Detailed resources guide stakeholders through integration, best practices, and utilizing the full potential of the system.
- Tested and reliable: Rigorous testing ensures a stable and dependable solution, with any issues addressed for optimal performance.
- Looking ahead: This project lays the groundwork for further innovation. Its scalable design allows adapting to evolving business needs and technology advancements.
- Impact: This success marks a significant step towards efficient business process management. By marrying BPMN.js's modeling with Neo4j's robust representation, the project contributes to the evolution of tools and methodologies for more agile, insightful, and interconnected process management in today's dynamic landscape.

Based on this work, we envisage the following areas of research as future work:

- Business Process Optimization[4]
- Automated Knowledge Graph Construction[5]
- Correctness Checking of BPMN Collaborations[6]

Bibliography

[1] G. Polančič and B. Orban, "A bpmn-based language for modeling corporate communications," *Computer Standards Interfaces*, vol. 65, pp. 45–60, 2019.

[2] P. Pullonen, J. Tom, R. Matulevičius, and A. Toots, "Privacy-enhanced bpmn: enabling data privacy analysis in business processes models," *Softw. Syst. Model.*, vol. 18, p. 3235–3264, dec 2019.

[3] S. Sakr and A. Awad, "A framework for querying graph-based business process models," in *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, (New York, NY, USA), p. 1297–1300, Association for Computing Machinery, 2010.

[4] F. Duran, C. Rocha, and G. Salaün, "Resource provisioning strategies for bpmn processes: Specification and analysis using maude," *Journal of Logical and Algebraic Methods in Programming*, vol. 123, p. 100711, 2021.

[5] S. Bachhofner, E. Kiesling, K. Revoredo, P. Waibel, and A. Polleres, "Automated process knowledge graph construction from bpmn models," (Berlin, Heidelberg), p. 32–47, Springer-Verlag, 2022.

[6] F. Corradini, A. Morichetta, A. Polini, B. Re, L. Rossi, and F. Tiezzi, "Correctness checking for bpmn collaborations with sub-processes," *Journal of Systems and Software*, vol. 166, p. 110594, 2020.

[7] M. von Rosing, S. White, F. Cummins, and H. de Man, "Business process model and notation—bpmn," in *The Complete Business Process Handbook* (M. von Rosing, A.-W. Scheer, and H. von Scheel, eds.), pp. 433–457, Boston: Morgan Kaufmann, 2015.

[8] "E2E Technologies Ltd., & 2013. bpmn 0.2.2." `https://www.npmjs.com/package/bpmn`.

[9] V. Goulart, V. Serra, M. Stamm, B. Mendes, bpmn.io Admin, N. Rehwaldt, pinussilvestrus, M. Barelkowski, P. Fromme, and M. Trumpf, "bpmn-moddle 8.0.1." `https://www.npmjs.com/package/bpmn-moddle`, 2022.

[10] J. Hansson and neo4j organization, "neo4j-driver 5.12.0." `https://www.npmjs.com/package/neo4j-driver`, 2023.

[11] M. Kubica, "xml2js 0.6.2." `https://www.npmjs.com/package/xml2js`, 2023.