UNIVERSITÀ DI PISA

Master Degree in Data Science
& Business Informatics

An Automatic, User-friendly Framework for Translating BPMN diagrams to Petri Nets

Supervisor:
Prof. Roberto Bruni

Candidate:
Andrea Napolitano

Department of Computer Science
Academic Year 2024/2025

Abstract

Business Process Model and Notation (BPMN) is a widely adopted standard for modeling business processes due to its graphical expressiveness and accessibility to both technical experts and business stakeholders. Despite its popularity, BPMN lacks a formal semantic foundation, which limits the ability to perform rigorous analysis and verification of process correctness. This gap hinders the application of formal methods in validating business workflows, especially in critical domains where correctness, consistency, and reliability are essential.

Petri nets, on the other hand, offer a solid mathematical formalism capable of modeling concurrency, synchronization, and resource sharing. They support formal verification techniques for properties such as soundness, liveness, reachability, and deadlock-freedom. By translating BPMN models into Petri nets, it becomes possible to combine the intuitive and communicative strength of BPMN with the analytical precision of formal methods.

This thesis proposes a novel web-based framework for the automated translation of BPMN diagrams into Petri nets. The system is designed to preserve the semantics of the original models while ensuring structural and behavioral correctness of the resulting nets. The implementation focuses on usability, offering features such as diagram visualization, model export, and integration with external analysis tools.

The system has been deployed to a web server which can be accessed through the website bpmn2petrinet.com and, the most curious lector, can find the source code available at github.com/BenjaNapo/bpmn-to-petri.

Ringraziamenti

Una domanda che mi pongo ultimamente è se le esperienze avrebbero lo stesso valore se non si condividessero con altre persone, una risposta non me la sono ancora data, però posso dire con certezza che questi due anni non sarebbero stati gli stessi se non li avessi condivisi con le persone con cui li ho condivisi.

Sarò banale, ma voglio iniziare ringraziando la mia famiglia: mamma, papà, sorella, Mattia e Zoe, per esserci stati in ogni momento e aver condiviso con me ogni mia ansia e traguardo. Ricordo ancora la mamma in lacrime quando son partito, convinti tutti che questi ultimi due anni sarebbero durati un'eternità, e invece eccoci qua. Nonostante la distanza, era come se fossero sempre con me a condividere ogni emozione e sapevano le parole giuste da dirmi per supportarmi quando ne avevo bisogno. Sia che si trattava di spiegarmi una nuova ricetta, risolvere problemi in giro per casa, darmi pareri sullo stile delle presentazioni o tranquillizzarmi quando mi autodiagnosticavo problemi al primo fastidio che mi sentivo addosso.
Sarò sdolcinato ma vi voglio bene <3

Successivamente ci sono due persone che tengo a ringraziare specialmente: Laba e Pigna.

Tutta questa esperienza non sarebbe mai esistita se Pigna non avesse deciso di venire a Pisa, destino o no, son felice che le cose siano andate così. Quello che ci tengo però a dire su di lui è il fatto che so essere un amico su cui posso contare sempre e comunque, un'ancora, che si tratti di condividere un momento di felicità o di risolvere l'ennesimo problema su qualche sito. Quella volta al mese in cui ci sentivamo, anche brevemente, con un suo "Ueee" che apriva i discorsi mi faceva uscire il sorriso. Sento sempre un'autenticità e purezza in quello che comunica, come se sapessi che posso affidarmi totalmente a lui che sarei in buone mani.

E ora veniamo a Laba, Laba... Compagno di almeno mille avventure, le cose da dire sarebbero tante, fortunatamente però ce le diciamo spesso anche senza nessuna occasione speciale, quindi questo sarà più un manifesto per fare sapere agli altri che bella persona sei. Sinceramente non potevo chiedere di meglio da questa convivenza, abbiamo vissuto così tante esperienze, concepito così tanti inner joke, superato momenti di ogni genere che se esco come persona nuova da qui è sicuramente anche grazie a te. Sei la persona che più assecondava le mie decisioni, come se avessi una fiducia totale nel fatto che io non potessi mai deluderti a prescindere dagli eventi. Ogni giorno si può dire che era speciale con te perché assieme sapevamo sempre dare un colore diverso alla giornata, che si trattava di cucinare qualche piatto particolare o di passare la sera a chiaccherare mentre facevi scorrege con le mani sul mio petto (non fate domande). Andrei avanti ancora per molto nel dire i pregi ma non voglio dilungarmi troppo, chi lo conosce un minimo riconosce subito che bella persona è.

Infine le persone da ringraziare sarebbero veramente tante, prima di venire qui non avrei mai creduto che le amicizie sarebbero state un punto fondamentale di questa esperienza, invece come sempre le mie aspettative non sono state rispettate e mi sono trovato a conoscere un sacco di bellissime persone e a condividere con ognuno bellissimi momenti. Quando tornavo su e parlavo di loro usando il termine "amici" sembrava quasi lo facessi con leggerezza, invece non è così, sento davvero di aver legato molto, a tal punto che se ci rivedessimo tra anni non sarebbe cambiato nulla.

I momenti più semplici sono quelli che sicuramente più mi resteranno, giusto per nominarne qualcuno: trascorrere le sere con Stefano a chiaccherare con una tisana in mano o passeggiando in giro, convincere Sandro a uscire a fumare per fare una pausa dallo studio, passare le sere a cenare a casa con Steffania, organizzare le serate internazionali per mangiare tutti assieme, giocare a scacchi con Andrea per poi piazzare un video di Malanga o, infine, uno dei momenti che più amavo di questa vita a Pisa, prendere il caffè con tutti dopo la mensa.

Enniente, questa esperienza mi ha fatto crescere più di quanto potessi immaginare e non posso che essere grato con tutti per averla condivisa con me.

Table of Contents

Chapter 1

Introduction

1.1 Context and Motivation

Business Process Model and Notation (BPMN) is a widely used standard for modeling business processes. Its syntactic simplicity and strong visual expressiveness make it an accessible tool for both technical analysts and non-specialized stakeholders. However, **BPMN** lacks a rigorous formal semantics, which makes it difficult to perform formal verification on process models and limits the possibilities for automated analysis.

In contrast, **Petri nets** represent a well-defined mathematical formalism, ideal for describing, simulating, and verifying the dynamic behavior of concurrent and distributed systems. They offer analytical tools to verify fundamental properties such as reachability, *soundness*, *liveness*, and the absence of *deadlocks*. Translating a **BPMN** model into a **Petri net** thus allows combining the readability and intuitiveness of the former with the rigor and precision of the latter, providing a solid foundation for the formal analysis of business processes.

1.2 Problem Statement

The need for this conversion is particularly relevant in industrial contexts, where the quality and reliability of processes have a direct impact on operational efficiency. An automated framework for translating **BPMN** into **Petri nets** can be extremely useful both for companies wishing to improve and validate their workflows and for researchers interested in workflow analysis tools based on formal models.

Once the equivalent Petri net has been obtained, it can be subjected to formal analysis through specialized tools such as **WoPeD** or **Woflan**, which allow verifying properties such as soundness, liveness, and the absence of deadlocks. In this way, it becomes possible to rigorously validate the original BPMN models, bridging the gap between graphical representation and analytical formalism.

1.3 Objectives and Scope

The objective of this thesis is the development of an automatic framework, accessible through a web interface, that allows the conversion of **BPMN** diagrams into **Petri nets**, preserving the semantics of the processes and ensuring structural and behavioral correctness. The system is designed to be intuitive for end users, offering advanced features for visualization, export, and analysis of the converted models.

Throughout the work, the theoretical foundations of **BPMN** and **Petri nets**, the motivations behind their integration, the design choices adopted for the development of the framework, and the strategies used to handle complex constructs such as *OR gateways* and timed *tasks* will be presented.

1.4 Existing Technologies

In the context of automatic translation from **BPMN** diagrams to **Petri nets**, various existing solutions have been analyzed to assess the possibility of integrating pre-existing tools into a user-friendly web interface. However, the conducted research revealed a lack of suitable solutions that provide results consistent with the methodology proposed in this work.

1.4.1 PM4Py

The **PM4Py** library is an open-source Python library for *process mining*, developed to support a wide range of algorithms and techniques for business process analysis. It enables the import, processing, and analysis of event data, facilitating tasks such as process discovery, conformance checking, and performance analysis.

One of the features offered by PM4Py is the conversion of **BPMN** models into **Petri nets**. However, during testing, several limitations of this function were observed. In particular, the conversion did not always produce accurate or complete results, indicating possible issues in the implementation or in handling certain complex BPMN constructs. For instance, discussions within the developer community highlighted difficulties in converting from Petri nets back to BPMN, suggesting the use of specific branches to address such issues [1].

These observations suggest that, although PM4Py is a powerful tool in the process mining landscape, its ability to convert BPMN models into Petri nets may require further improvements to ensure reliable and accurate conversion.

[1] Issue #60: Problems with conversion from Petri nets to BPMN. `https://github.com/pm4py/pm4py-source/issues/60`

1.4.2 BPMN 2 Petri Nets Transformation

Another analyzed solution is the Java library **BPMN 2 Petri Nets Transformation**, available on GitHub [1]. This project offers a transformation from **BPMN** models to **Petri nets** through a web application. However, some issues were identified. Firstly, the provided link to access the online application (`http://islab1.ieis.tue.nl:6411/Transformation/`) is inactive, preventing direct testing of the tool. Moreover, the last update of the repository dates back to about ten years ago, suggesting potential incompatibility with current technologies and standards.

1.4.3 ProM

ProM is an open-source framework developed in Java, widely used in the academic community for *process mining* activities. The framework is based on a modular plug-in architecture that allows the addition of customized functionalities, including tools for the discovery, verification, and conversion of process models.

Among the available plug-ins, ProM includes tools for converting **BPMN** models into **Petri nets**, such as *"Convert BPMN diagram to Petri net (control-flow)"* and *"Convert BPMN diagram to Data Petri net"*, included in the `BPMNConversions` package. However, despite the availability of these features, the practical use of **ProM** in a web context is complex: the framework is designed for desktop use and does not offer direct support for integration into modern web interfaces, and the Petri nets generated by these plug-ins are often unnecessarily complex and presented with a layout that is difficult to interpret and analyze.

Moreover, as highlighted in the literature, the development of **ProM** initially seemed to be mainly oriented toward conversion from **Petri nets** to **BPMN**, rather than the other way around [2].

1.5 Decision to Develop In-House

Given the limitations of existing solutions, it was necessary to evaluate the most effective approach to achieve the intended goals. The choice was between:

1. **Developing a system from scratch**: Creating a new application that fully meets the functional and usability requirements, ensuring accurate translation and easy integration into a modern web interface.

2. **Integrating existing solutions**: Adapting PM4Py by correcting the Python code related to the conversion and publishing it on a server to expose HTTP-accessible APIs.

Following the conducted analysis, the most robust and sustainable option was found to be the development of an entirely new system. This choice ensured full control over the implementation of the conversion logic, allowing the creation of a tailored solution. The computational lightness of the conversion process also made it possible to perform the execution entirely locally, eliminating the need to interface with external APIs and simplifying the overall system architecture.

Structure of the Thesis

The structure of this thesis is organized to guide the reader through the entire development and implementation process of the proposed solution.

- **Chapter 1: Introduction** - This chapter provides a general overview of the project, outlining the objectives and motivations behind the research.
- **Chapter 2: BPMN and Petri Nets** - This chapter explores the theoretical concepts related to BPMN (Business Process Model and Notation) and Petri nets, focusing on their interrelationship and the methodology for translating the former into the latter.
- **Chapter 3: Solution Design** - This chapter describes the architecture of the developed system, focusing on the technological choices and the design of the conversion framework.
- **Chapter 4: Translation Pipeline** - This chapter examines in detail the stages of the conversion process, from reading the BPMN diagram to the export of the Petri net.
- **Chapter 5: Additional Features** - This chapter delves into the advanced features of the system, such as gateway handling and configuration customization.
- **Chapter 6: Results and Testing** - This chapter presents the results obtained from the system through practical tests, highlighting the performance and reliability of the solution.
- **Chapter 7: Conclusions** - This chapter summarizes the main findings of the thesis and suggests potential future developments for the project.

Chapter 2

BPMN & Petri Net

Business process modeling relies on various formalisms, each with specific characteristics and purposes. Among these, **Business Process Model and Notation (BPMN)** and **Petri nets** represent two complementary approaches: the former focuses on communicative clarity and widespread industrial adoption, while the latter emphasizes formal precision and behavioral analysis. This chapter introduces both formalisms, analyzing their main components, their roles in process modeling, and why combining them is an effective strategy to merge usability with analytical rigor. For readers interested in delving deeper into the concepts, the works of Weske (2019) [3] and Dumas (2018) [4] provide an extensive overview of the subject, while the foundational text by van der Aalst and van Hee [5] offers a detailed treatment of workflow modeling, analysis, and system support.

2.1 BPMN Elements

Business Process Model and Notation (BPMN) [6] is a graphical standard used to represent business processes clearly and structurally. It defines a set of distinct elements, grouped into categories, each with a specific role within the process. As discussed in [7], BPMN plays a central role in the behavioural modelling of business processes, providing an expressive yet structured language for capturing control flow and interaction patterns.

This section outlines the main element groups that compose a **BPMN** model, highlighting their characteristics and functions.

2.1.1 Flow Objects

Flow Objects are the core elements that describe process behavior. They include:

- **Events**: represent occurrences that affect the process flow. They can signal the start, interruption, or completion of an activity. Events are essential in BPMN modeling as they define the dynamic behavior of the process. Events are divided into:

  - *Start Event*: indicates the starting point of the process. It is the first element triggered and can take several forms, such as a simple event, timer, message-based, or signal-based.
  - *Intermediate Event*: occurs during the execution of the process. It can serve different purposes, such as receiving a message, waiting for a timer, or handling exceptions. It may be:
    * **Catching**: waits for an external event before proceeding.
    * **Throwing**: generates an event to be received by another part of the process.
  - *End Event*: marks the end of the process or a subprocess. Variants include standard exit, message-based exit, or termination of the entire flow.

- **Activities**: represent actions performed in the process. These include:
  - *Task*: an atomic activity that cannot be further decomposed.
  - *Sub-Process*: a compound activity that encapsulates a detailed process.
  - *Call Activity*: an activity that invokes another process.

- **Gateways**: decision points used to control the process flow, defining divergence and convergence. Each gateway operates according to specific rules. Key types include:
  - **Exclusive Gateway (XOR)**: Selects a single path among alternatives. On split, a condition determines which outgoing path to take. On join, it waits for one incoming flow before proceeding.
  - **Parallel Gateway (AND)**: Used for simultaneous execution. On split, all outgoing paths are triggered at once. On join, it waits for all incoming flows to complete before proceeding.
  - **Inclusive Gateway (OR)**: Enables one or more outgoing paths based on conditions. On split, multiple paths may be activated concurrently. On join, it waits for all triggered paths to complete.
  - **Event-based Gateway**: Determines the path based on external events instead of conditions. The process waits for one of the defined events to occur and proceeds accordingly. Events may include signals, timers, or messages.
  - **Complex Gateway**: Allows for sophisticated behavior by combining activation rules, enabling complex decision logic. For example, it may require a specific subset of incoming flows to proceed.

2.1.2 Connecting Objects

Connecting Objects link different components of the process and include:

- **Sequence Flow**: shows the execution order between two elements.
- **Message Flow**: represents communication between separate entities (e.g., different participants).
- **Association**: links additional information such as annotations or artifacts to process elements.

2.1.3 Swimlanes

Swimlanes organize responsibilities and task distribution within a process. They include:

- **Pool**: represents an independent entity or participant.
- **Lane**: subdivides a pool to specify roles or responsibilities.

2.1.4 Artifacts

Artifacts provide additional information to support understanding of the model without affecting the process flow. Key artifacts include:

- **Data Objects**: represent data used or produced in the process.
- **Groups**: cluster process elements for readability.
- **Annotations**: add comments or descriptive notes.

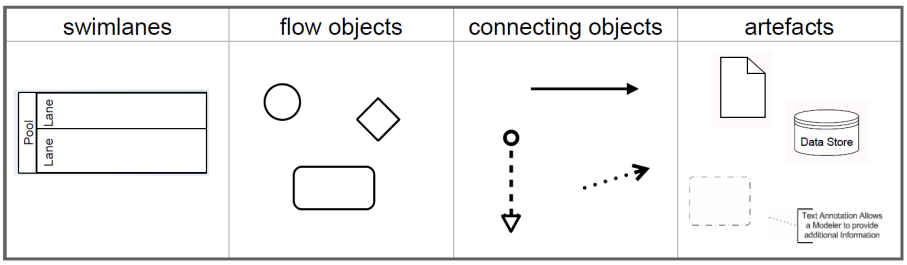

Figure 2.1: Basic BPMN elements

2.1.5 Advanced Constructs

Beyond basic elements, BPMN includes advanced constructs for modeling complex interactions and error handling:

- **Transaction**: a special type of *Sub-Process* representing an atomic unit of work. All contained activities must complete successfully; otherwise, the entire transaction is rolled back.
- **Compensation**: a mechanism to reverse completed activities when an error occurs later in the process. It's useful for handling corrective actions such as refunds or cancellations.

2.1.6 Examples of BPMN Diagrams

In this section, we present two examples of BPMN diagrams: a simple process diagram and a collaboration diagram. These examples will help to illustrate the core elements of BPMN and how they are visually represented.

Single Process Example

The first example demonstrates a simple BPMN process. This diagram represents an order fulfillment process in BPMN. It includes events, tasks, and gateways to depict the flow from receiving a purchase order to fulfilling the order. In particular, "Purchase order received" is the start event and is followed by a XOR gateway that may lead directly to the task "reject order" or to the activation of a series of tasks that involve the parallel execution of shipment and invoicing activities (enclosed in a parallel block delimited by AND gateways), until the final event "order fulfilled" is reached.

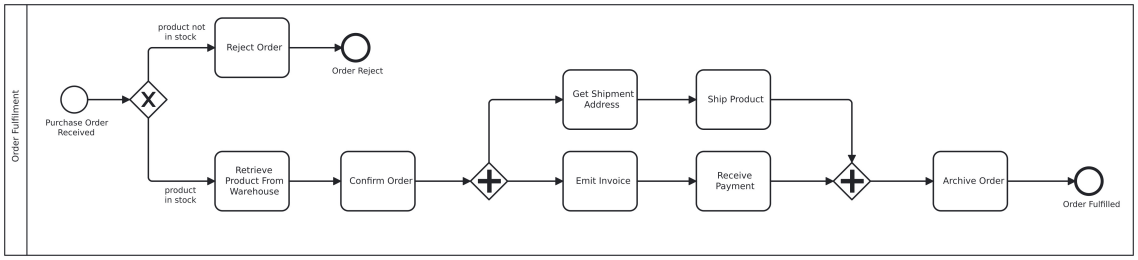

Figure 2.2: Simple BPMN Process Example

Collaboration Example

The second example illustrates a BPMN collaboration diagram, which models interactions between a travel agency and a tourist, capturing the flow of interactions during the planning and confirmation of a trip. Notably, the diagram highlights

the use of decorations for sending and receiving messages, emphasizing the communication between the two participants. Additionally, the diagram incorporates an Event-Based Gateway, which introduces decision points where the flow diverges based on the occurrence of specific events, such as receiving a response or confirmation from the tourist.

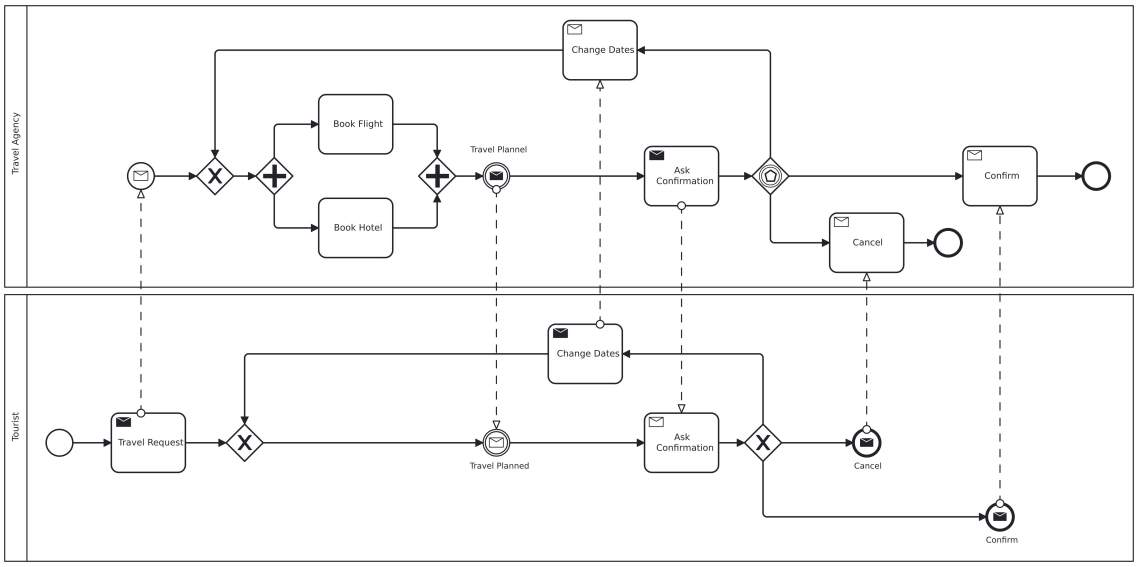

Figure 2.3: BPMN Collaboration Example

2.2 Petri Net Elements

Petri nets are a mathematical tool for modeling and analyzing concurrent systems, suitable for representing processes with synchronized events and shared resources. Originally introduced by Carl Adam Petri in his doctoral thesis [8], they have become a cornerstone of formal modeling. Reisig's introductory text [9] provides a comprehensive overview of their theoretical foundations, while his later works [10] and [11] focus on modeling techniques, analysis methods, and practical case studies. Additionally, the work of van der Aalst and Stahl [12] offers a Petri net-oriented approach to modeling business processes, bridging the gap between theoretical formalisms and real-world applications.

A Petri net is formally defined as a tuple $N = (P, T, F)$, where:

- P is a finite set of **places** (circles), representing system states.
- T is a finite set of **transitions** (squares), representing events that may change the state.
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of **arcs** (arrows), connecting places and transitions to define system dynamics.

2.2.1 Markings and System Definition

Marking: A marking is a function $M : P \to \mathbb{N}$ that assigns a non-negative integer to each place p, indicating the number of tokens it contains. The current state of the system is represented by a marking, often viewed as a vector in $\mathbb{N}^{|P|}$. A place p is said to be **marked** if $M(p) > 0$, and **unmarked** otherwise.

A **Petri net system** is defined as the pair (N, M_0), where M_0 is the **initial marking** from which execution begins. The behavior of the system is captured by the evolution of markings through the firing of transitions.

A transition is said to be **enabled** when all of its input places contain at least one token. Once enabled, the transition may **fire**, meaning it consumes one token from each input place and produces one token in each output place. This firing operation changes the marking of the net, representing the progression of the system's state.

In other words, transitions simulate events or actions: they can only occur when specific conditions (token availability) are met, and their occurrence leads to a new configuration of tokens across the places. This mechanism captures both causality and concurrency in the modeled process.

Pre-set and Post-set

For any node (whether a place or transition), the **pre-set** and **post-set** describe the other nodes that are connected to it, formally:

$$\bullet t = \{p \mid (p, t) \in F\} = \text{the set of input places of } t$$

$$t \bullet = \{p \mid (p, t) \in F\} = \text{the set of output places of } t$$

$$\bullet p = \{t \mid (t, p) \in F\} = \text{the set of input transitions of } p$$

$$p \bullet = \{t \mid (p, t) \in F\} = \text{the set of output transitions of } p$$

2.2.2 Workflow Net

A **Workflow Net** is a special class of Petri nets used to model workflows or processes. Formally, a net (P, T, F) is a workflow net if:

1. There exists a (unique) initial place $i \in P$ such that $\bullet i = \emptyset$
2. There exists a (unique) final place $o \in P$ such that $o \bullet = \emptyset$
3. Every node of the net belongs to an oriented path from i to o

The Workflow Net thus guarantees that all tokens are processed in a well-defined sequence, and it captures the control flow of the modeled process.

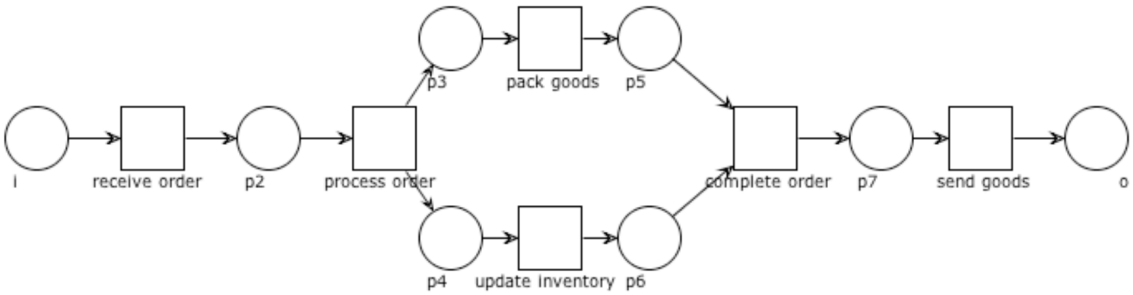

Figure 2.4: Workflow Net Example

The diagram in Figure 2.4 represents a **Workflow Net** (WF-net), modeling a simple order processing workflow. The net has a single initial place and a single final place, with all nodes connected by directed paths, ensuring the proper sequence and completion of tasks. This structure captures the flow of control from start to finish without leaving unprocessed states.

2.2.3 Visual Enhancements: Decorators

In some Petri net representations, **decorators** are used as visual aids to clarify the logical role of certain transitions, especially in the presence of complex control-flow structures. While they do not affect the semantics or execution of the net, decorators help to highlight the behavioral intention behind specific modeling choices.

There are four common types of decorators, each associated with a particular control construct:

- **XOR Split**: applied to transitions that represent exclusive choices. Only one of the outgoing arcs will be activated upon firing. The decorator indicates that a decision is being made.

- **XOR Join**: used when multiple incoming flows may arrive, but only one is expected at a time. The decorator marks that the transition merges alternative paths, without requiring synchronization.

- **AND Split**: identifies transitions that initiate multiple flows in parallel. All outgoing arcs are simultaneously activated, modeling concurrent execution.

- **AND Join**: indicates synchronization points where all incoming tokens must be present before the transition can fire. This decorator is used to represent the convergence of parallel branches.

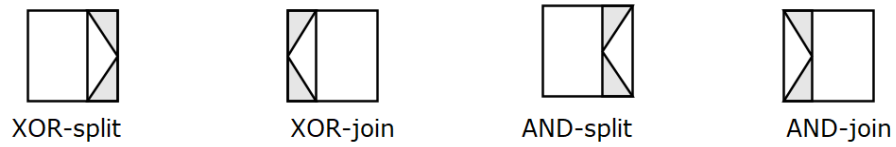

Figure 2.5: Simple decorators in WoPed

Some Petri net modeling tools, such as WoPeD, provide predefined combinations of **join** and **split** behaviors, which can be selected directly from the graphical interface. Examples include elements like `XOR-join-split`, `XOR-join-AND-split`, `AND-join-split`, and `AND-join-XOR-split`. These constructs allow the representation of transitions that simultaneously act as convergence and divergence points, using different logic for incoming and outgoing arcs (e.g., XOR on input, AND on output).

In addition, some tools offer support for modeling `Subprocesses`, allowing the grouping of related activities under a single hierarchical component. This feature is useful for abstracting parts of the model and improving its overall readability.

2.2.4 Properties of Petri Nets

Petri nets can be analyzed through several properties that ensure model correctness and expected behavior:

- **Liveness**: A transition is live if, from any reachable marking, it is always possible to eventually fire it in the future.

$$\forall M \in [M_0>, \exists M' \in [M> \text{ such that } M' \xrightarrow{t}$$

- **Place-liveness**: A place is live if, from any reachable marking, there exists a future marking where the place contains at least one token.

$$\forall M \in [M_0>, \exists M' \in [M> \text{ such that } M'(p) > 0$$

- **Dead transition**: A transition is dead if there is no reachable marking from which it can be fired.

$$\forall M' \in [M>, \quad M' \not\xrightarrow{t}$$

- **Dead place**: A place is dead if, in all reachable markings, it never contains any tokens.

$$\forall M' \in [M>, \quad M'(p) = 0$$

- **Deadlock-freedom**: A Petri net is deadlock-free if, from every reachable marking, there is always at least one transition that can fire.

$$\forall M \in [M_0>, \exists t \in T \text{ such that } M \xrightarrow{t}$$

- **Boundedness**: A net is bounded if the number of tokens in each place never exceeds a finite value k, in all reachable markings.

$$\forall M \in [M_0>, \quad M(p) \leq k$$

- **Safeness**: A net is safe if it is 1-bounded, meaning that no place can contain more than one token in any reachable marking.

- **Soundness (Workflow Nets)**: A workflow net is sound if it satisfies three properties:

  1. **No dead task**: Every transition can eventually be executed in some reachable marking.

$$\forall t \in T, \exists M \in [i> \text{ such that } M \xrightarrow{t}$$

  2. **Option to complete**: From any reachable marking, it is possible to reach the final marking.

$$\forall M \in [i>, \exists M' \in [M> \text{ such that } M'(o) \geq 1$$

  3. **Proper completion**: If the final place is marked, then no other places are marked (the process terminates cleanly).

$$\forall M \in [i>, M(o) > 0 \Rightarrow M = o$$

2.3 Translation

Translating a **BPMN** diagram into a **Petri net** is non-trivial. It requires converting intuitive graphical elements into rigorous mathematical structures. Each BPMN construct (e.g., **activities, gateways, and events**) must be mapped to corresponding **places, transitions, and arcs**, preserving process semantics.

The main challenges are:

1. **Accurate mapping of BPMN elements** to Petri net structures;
2. **Preservation of process behavior**, ensuring transitions reflect workflow logic;
3. **Analysis of the resulting structure**, verifying it is **well-formed and analyzable**.

2.3.1 Constraints & Assumptions

Specific structural constraints and assumptions are imposed on the **BPMN** diagram to ensure a correct translation:

1. A *gateway* cannot simultaneously have more than one incoming and more than one outgoing arc. This ensures a clear split/join distinction.
2. Each **task** must have exactly one *message flow* incoming and outgoing. This guarantees correct communication modeling.
3. An **OR Split** gateway is translated as an **XOR Split** followed by several **AND Splits** to represent all possible outgoing combinations. The **OR Join** is reversed: **AND Joins** followed by an **XOR Join**. However, this strategy may compromise the resulting Petri net's *soundness*.
4. **Transactions** and **compensations** are not supported. These constructs could introduce ambiguous or hard-to-model behavior in the Petri net.
5. **Complex Gateways are not supported**: due to their arbitrary and undefined semantics, their conversion would require non-deterministic rules, undermining correctness.

2.3.2 Translation Steps

The translation process follows structured steps:

Step 1: Flow Conversion (Sequence Flow and Message Flow)

Each **sequence flow** and **message flow** is converted into a **place**, representing a process state. This ensures all connections between elements are preserved in the Petri net.

Step 2: Conversion of Flow Elements (Flow Objects)

Each behavioral element is translated as follows:

- **Events & Activities**: translated to **transitions**.
- **Gateways**:
  - **XOR**: split—intermediate place with transitions per path; join—transitions converge into a single place.
  - **AND**: split—one transition to multiple places; join—multiple places to one synchronized transition.
  - **OR**: generate all non-empty subsets; transitions for each subset manage token routing.

– **Event-based**: one place with multiple transitions for each event; the first to trigger proceeds.

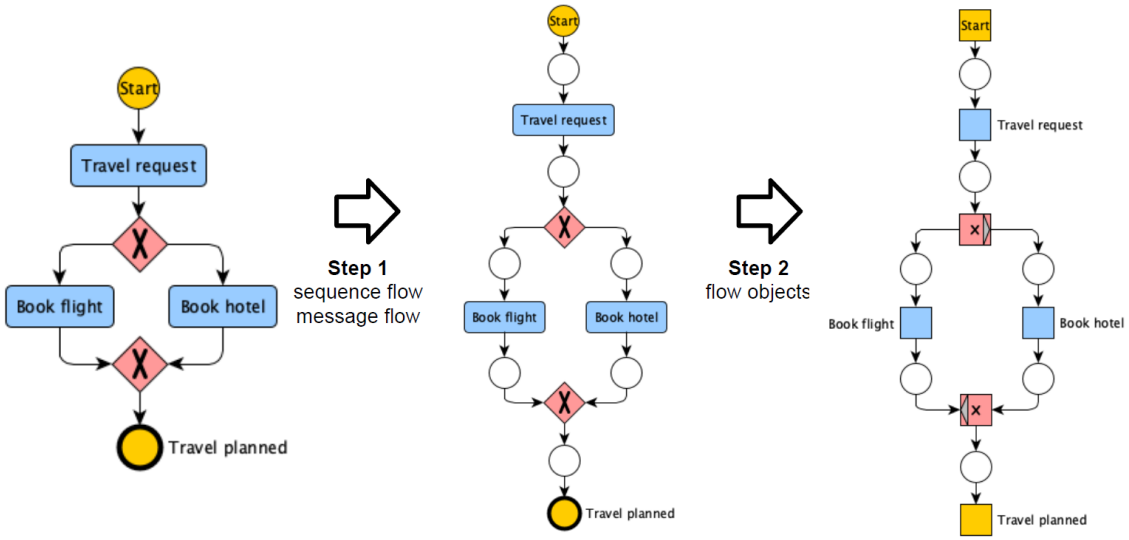

Figure 2.6: Step 1 and 2 of the translation process

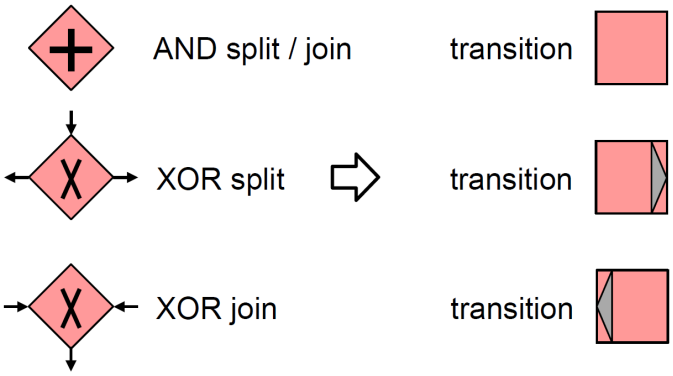

Figure 2.7: AND & XOR gateways mapping from BPMN to Petri Net

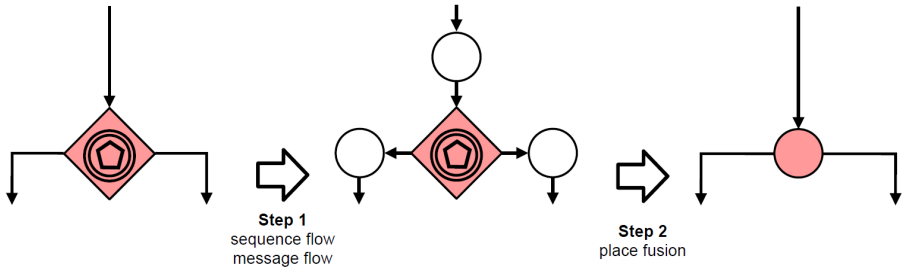

Figure 2.8: Event-Based gateway conversion steps

Step 3: Initial and Final Conditions

To coordinate the execution of parallel processes, two special places are added: one initial and one final. These are connected to the start and end places of each lane via an AND transition, ensuring a synchronized start and termination of all flows.

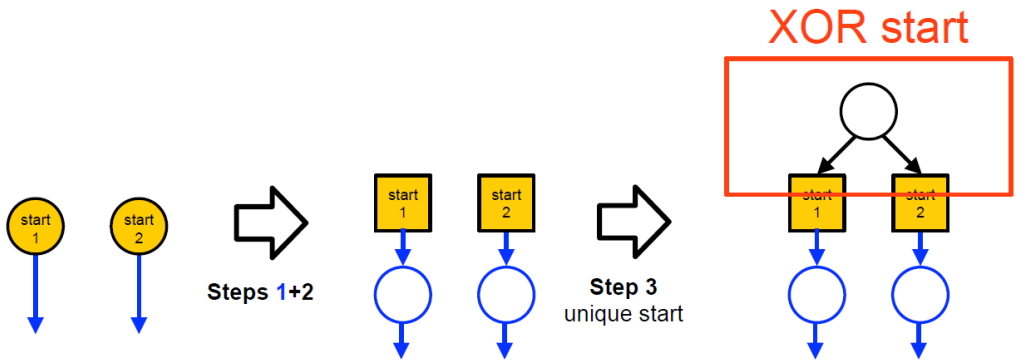

Figure 2.9: Addition of an unique starting place

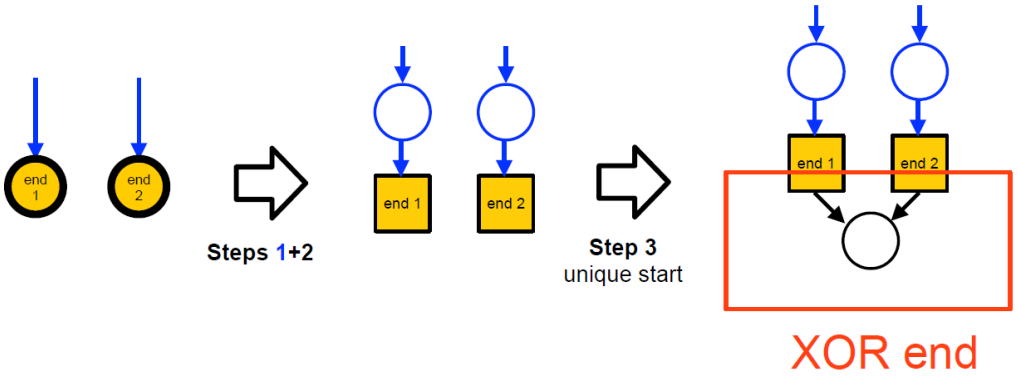

Figure 2.10: Addition of an unique ending place

2.3.3 Translation of BPMN Examples to Petri Nets

In this section, we demonstrate the translation of BPMN examples into Petri nets using the tool developed for this purpose. The diagrams show two representative cases of workflow processes.

The conversion demonstrates how Petri nets can be used as an effective means of representing and analyzing the behavior of business processes defined in BPMN, with the added benefit of formal verification and analysis capabilities.

The network shown in Figure 2.11 illustrates the translation of the simple diagram presented in Figure 2.2. As can be observed, the tasks and activities have been translated into transitions, while the edges are represented as places.

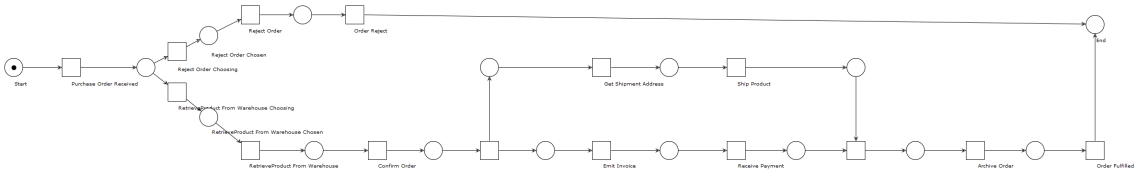

Figure 2.11: Petri Net Translation of the BPMN Order Processing Workflow.

In this second example, shown in Figure 2.12, we present the Petri net translation of the collaboration example depicted in Figure 2.3. As observed, in this case, the message flows are also transformed into places, ensuring that the incoming transitions require tokens from both the sequence flow and the message flow to be fireable. Additionally, since there are two pools in this example, synchronization has been achieved by unifying the entire process through an initial and final transition, both connected to an initial and final place.

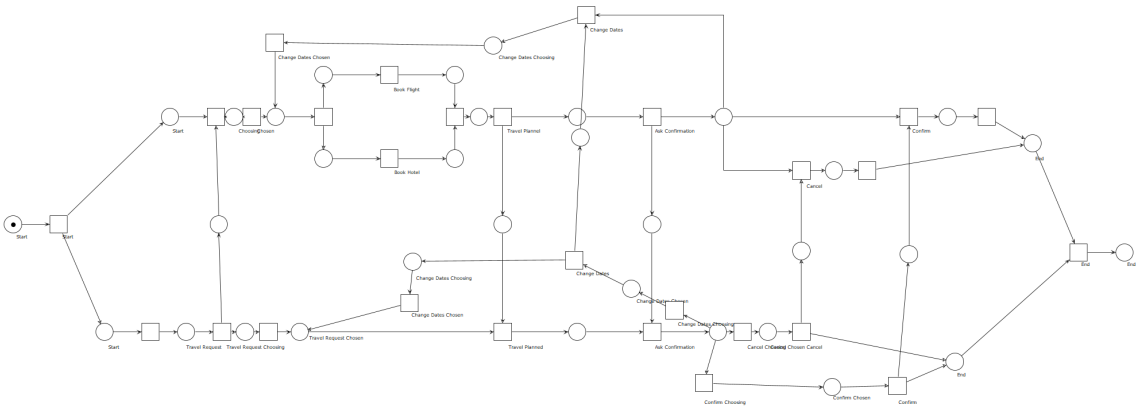

Figure 2.12: Petri Net Translation of the BPMN Travel Agency Process.

Chapter 3

Solution Design

The design of the solution represents a crucial phase to ensure an effective and functional implementation of the conversion from **BPMN** to **Petri nets**. This section analyzes the adopted design choices, with particular attention to the evaluation of available technologies, the selection of tools used, the requirement analysis, and the system structure through the class diagram.

3.1 Technologies & Architecture

This section outlines the technologies and tools used to develop the system, detailing the programming languages, frameworks, libraries, and tools that form the foundation of the project. The architecture of the system was carefully designed to ensure efficient performance, easy accessibility, and seamless integration, enabling the conversion of BPMN diagrams into Petri nets.

3.1.1 Programming (& non) Languages

The entire system was developed using only **JavaScript**, creating a dedicated module called `bpmn2petri`, responsible for the automatic conversion of **BPMN** diagrams into **Petri Nets**. The user interface was built using **HTML** and **CSS**, ensuring a user-friendly experience directly accessible via browser without requiring additional installations.

Both **BPMN** diagrams and **Petri nets** are represented using **XML**-based formats. In particular:

- **BPMN**: files with the `.bpmn` extension are **XML** documents compliant with the **BPMN 2.0** specification, defining business process structures through a standardized language.
- **PNML (Petri Net Markup Language)**: Petri nets are saved as `.pnml` files, also based on **XML**. This format is designed for the formal representation of Petri nets and is compatible with existing analysis and simulation tools.

3.1.2 Frameworks and Libraries

For BPMN diagram management and visualization, the **bpmn.js** library was used. This **JavaScript**-based library provides full support for modeling, visualizing, and interacting with BPMN diagrams in a web environment.

Integrating **bpmn.js** allowed for:

- Loading and parsing `.bpmn` files in **XML** format.
- Visualizing BPMN diagrams within the web interface.
- Allowing users to navigate the model.

3.1.3 Development and Integration Tools

Git was used for version control, with repositories hosted on **GitHub**. During the development phase, the project was maintained in a private repository managed via **GitHub Desktop**, which facilitated version tracking and code synchronization. The project was initially developed in a private repository named `bpmn-to-petri-pvt`. Upon completion of the first stable release, a new public repository called `bpmn-to-petri` was created and published at https://github.com/BenjaNapo/bpmn-to-petri, making the project openly available to the community for future enhancements and contributions.

Development was conducted using **Visual Studio Code (VS Code)**, a code editor that, with the help of extensions, ensured an efficient workflow.

For real-time testing and interface preview, the **Live Server** extension in **VS Code** was used, allowing instant testing without the need for external server setup.

To verify the correctness of the generated Petri nets, the **WoPeD** tool was used, enabling import of PNML files for structural checks and simulations.

3.1.4 Hosting and Deployment

To make the system easily accessible, the domain **bpmn2petrinet.com** was purchased, through which the web application will be publicly available. The site is hosted using the **shared hosting** service provided by **Hostinger**, offering a good balance between cost, performance, and ease of management.

3.2 Requirements Analysis

Requirements analysis is a fundamental phase in system design, as it defines essential features, technical specifications, and user needs. This section identifies and classifies the project requirements according to their nature and impact on the system architecture.

3.2.1 Functional Requirements

The **bpmn2petrinet.com** system is designed to provide a web-based service that allows users to automatically and efficiently convert BPMN diagrams into Petri nets. Functional requirements specify the operations the system must perform to ensure proper functioning of the conversion process.

1. **Upload BPMN diagrams**: users must be able to upload `.bpmn` files through the web interface, supporting BPMN 2.0 standard.
2. **BPMN file parsing**: the system must interpret the XML structure and correctly identify its elements.
3. **BPMN visualization**: the uploaded BPMN diagram must be rendered to allow structural verification by the user.
4. **Automatic conversion to Petri net**: the system must convert the BPMN diagram into a Petri net, applying conversion rules that ensure structural and behavioral correctness.
5. **Petri net visualization**: the generated Petri net must be displayed graphically, allowing side-by-side comparison with the original BPMN.
6. **Petri net export**: users must be able to download the generated Petri net in various formats, with advanced export options:
    - **Classic export**: full Petri net derived from the original BPMN.
    - **Export by pool**: each BPMN pool is converted into a separate Petri net, excluding message flows.
    - **Graphviz format export**: the Petri net is exported in a **Graphviz**-compatible format for external graphical rendering.
7. **Error handling**: the system must detect unsupported BPMN elements or invalid configurations and return clear error messages.
8. **Conversion options configuration**: users must be able to configure conversion parameters through a settings panel, including:
    - **Apply Decorators**: apply decorators to resulting gateways.
    - **Collapse XOR**: compact XOR by avoiding intermediate transitions.
    - **Timed Tasks**: represent tasks with separate start and end transitions.
    - **Node Size**: adjust node size in the Petri net.
    - **Flow Scaling**: modify flow connection scaling factor.
    - **Graphviz Text**: set text position in Graphviz renderings.

9. **View modes**: the system must offer different display modes for analyzing and comparing the BPMN diagram and generated Petri net:
    - **Vertical split**: BPMN above, Petri net below.
    - **Horizontal split**: BPMN and Petri net side by side.
    - **BPMN only**: show only the BPMN diagram.
    - **Petri Net only**: show only the Petri net.
10. **Batch Mode**: users must be able to upload and process multiple BPMN files at once. This includes:
    - **Individual visualization**: each BPMN diagram is displayed with its corresponding Petri net.
    - **Single download**: each converted Petri net can be downloaded separately as a `.pnml` file.
    - **Collective download**: all Petri nets can be downloaded at once with a single click.

3.2.2 Non-Functional Requirements

Non-functional requirements describe qualitative aspects of the system, such as usability, performance, and security.

1. **User-friendly interface**: the interface must be simple and intuitive, allowing users to perform conversions easily.
2. **Processing speed**: conversions must be completed in a reasonable time, even for complex diagrams.
3. **Reliability**: the system must deliver consistent results for each operation, minimizing errors and unpredictable behavior.
4. **Maintainability and updatability**: the code must be modular to support future updates and improvements.
5. **Reliable hosting**: the site must be continuously available, with infrastructure that minimizes downtime.
6. **Browser compatibility**: the system must run correctly on all major browsers (Chrome, Firefox, Safari, Edge) without additional plugins.
7. **Client-side processing**: the conversion must take place in the user's browser without sending data to an external server.

3.3 Class Diagram

The system for translating BPMN diagrams into Petri Nets is structured as a modular pipeline, orchestrated by a central **Event Manager**. Each module is responsible for a distinct stage in the translation process, ensuring separation of concerns, reusability, and scalability. The architecture is depicted in the following diagram:

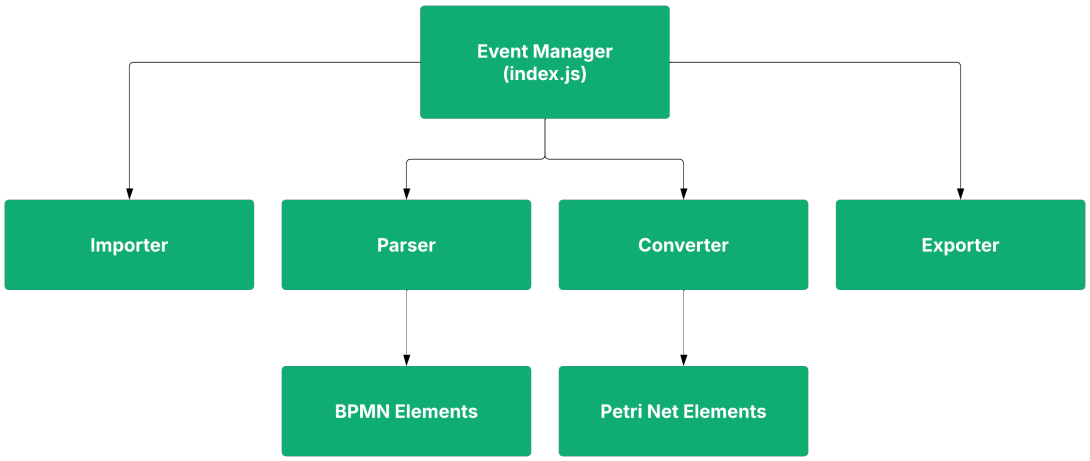

Figure 3.1: High Level Class Diagram

The system is organized into a modular architecture, with each component responsible for a distinct phase of the translation process from BPMN to Petri Nets. At its core lies the **Event Manager** (implemented in `index.js`), which orchestrates the overall workflow, handles user interactions, and triggers the execution of specific modules such as import, parse, convert, and export. The process begins with the **Importer**, which loads external `.bpmn` files and prepares their contents for parsing. The **Parser** then processes the raw BPMN data and generates a structured internal representation, organizing elements and their relationships into manipulable JavaScript objects.

This representation is passed to the **Converter**, the core logic engine of the system, which applies formal transformation rules to map BPMN constructs into Petri Net components while preserving behavioral semantics. The converter handles tasks, events, and gateways by converting them into corresponding Petri Net places and transitions using dedicated handlers, and manages intermediate mappings to ensure structural consistency.

Once the translation is complete, the **Exporter** serializes the resulting Petri Net into both `.pnml` and `.dot` formats. Underlying this pipeline are two key model definition files: the **BPMN Elements** (`bpmn.js`), which contains classes such as `BPMN`, `Process`, `BPMNNode`, and `Flow` for representing BPMN diagrams, and the **Petri Net Elements** (`petrinet.js`), which includes classes like `PetriNet`, `Node` and `Arc` for building the resulting Petri Net structure.

3.4 Implementation

The entire system is developed as a single HTML page that loads everything at startup. The project is structured into three main components:

- **System display:** HTML is used for the frontend structure, and CSS is used to style graphical elements.
- **Event handling:** a file named *index.js* handles button events using **jQuery**, a library that simplifies DOM manipulation and event handling.
- **bpmn2petrinet module:** a separate folder contains all the files responsible for implementing the system's core functionalities.

Below is the final file structure:

- **assets/** folder for images and icons used on the site.
- **css/** folder containing the CSS files for styling.
- **bpmn_js/** module used to display uploaded BPMN diagrams.
- **src/** main source code folder
  - **bpmn2petri/** module responsible for all core functionalities.
  - **index.js** links modules and handles execution of the conversion pipeline.
- **index.html** defines the system structure and links styles and logic.

Chapter 4

Translation Pipeline

The conversion process from BPMN to Petri net is structured as a multi-phase pipeline that ensures a systematic and modular transformation. It begins with the import and parsing of the BPMN file, proceeds through semantic interpretation and structural translation, and concludes with the graphical rendering and export of the resulting Petri net.

This chapter provides a detailed overview of each stage in the pipeline, highlighting the key operations involved, the data structures used, and the intermediate representations that allow the system to preserve both the structure and semantics of the original model.

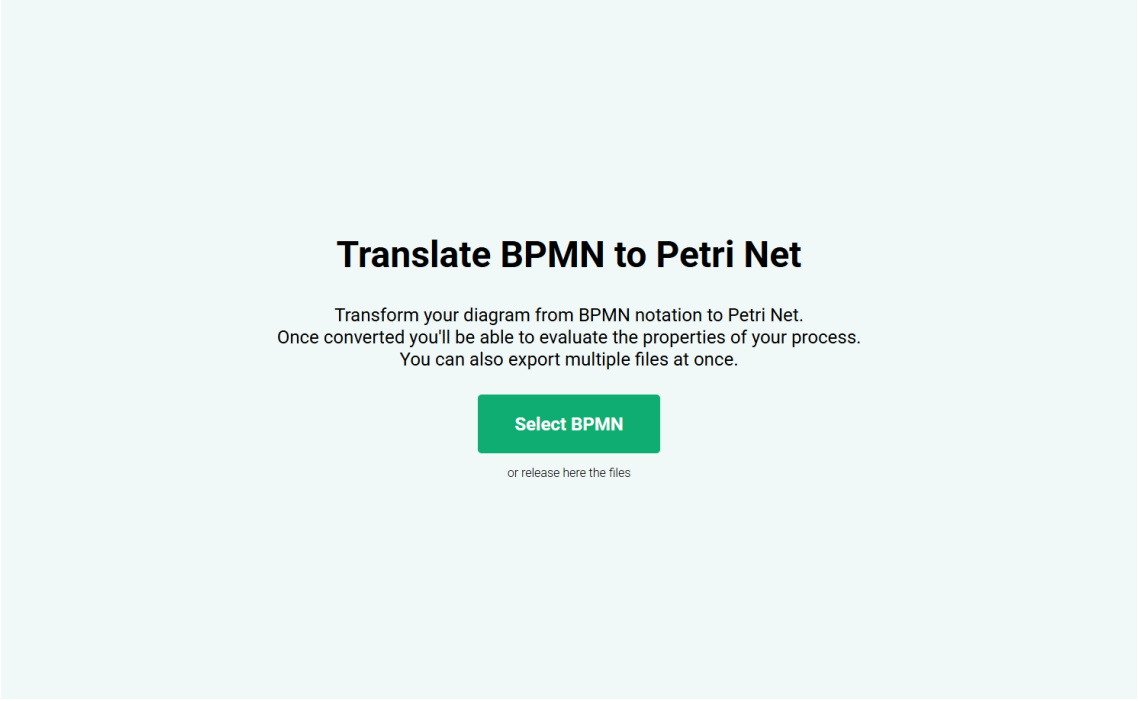

Figure 4.1: Homepage of the system

4.1 Importer

The first stage of the conversion pipeline consists of importing the content of the BPMN file in XML format. To handle this, a dedicated module named `importer.js` was developed, which encapsulates all logic for loading and parsing BPMN diagrams. The core component is the `Importer` class, which exposes two main methods: one for importing content from a `File` object, and another for parsing directly from a `string`.

The separation of these two methods allows the system to support both file uploads and programmatic parsing, ensuring flexibility in different usage contexts (e.g., batch mode, API usage, or direct in-browser editing).

Given the presence of various notational styles across BPMN editors, the system also performs a normalization step. Specifically, if the `removePrefix` flag is enabled (default behavior), the importer automatically strips any namespace prefixes from tag names (e.g., `bpmn:task` becomes `task`). This ensures that all diagrams are processed uniformly, regardless of the tool used for their creation.

Algorithm 1 Importing a BPMN/XML file

Require: BPMN file to be imported
Ensure: Parsed XML Document
1: SupportedTypes ← {".txt", ".bpmn", ".xml"}
2: **if** file.extension ∈ SupportedTypes **then**
3: Read file content as text
4: **if** RemovePrefix is enabled **then**
5: Remove namespace prefixes from XML tags
6: **end if**
7: XML ← Parse BPMN content as XML **return** XML
8: **else**
9: Raise error "`File not supported!`"
10: **end if**

Internally, the importer uses a `DOMParser` to parse the BPMN content and stores both the original and the normalized version. This design also allows advanced debugging or fallback strategies in case issues arise during conversion.

The `importer.js` module is designed to be reusable and modular, forming the entry point for all conversion operations that follow.

4.2 Parser

The second phase of the pipeline is dedicated to parsing the previously imported XML document and transforming it into a structured BPMN object model. This model includes a collection of processes (one for each pool), each containing the

internal nodes and control-flow arcs, as well as a separate set of message flows that describe inter-pool communications.

The `Parser` class is initialized with an XML document and performs the following main operations during construction:

1. **Parsing graphical layout of elements**: the `parseShapesLayouts()` method extracts the position and dimensions of BPMN nodes using the `BPMNShape` and `Bounds` tags. This information is essential to maintain the original layout in the final Petri net.

2. **Parsing edge paths**: the `parseEdgesLayouts()` method collects waypoints from `BPMNEdge` tags, storing the polyline representation of each arc to preserve graphical fidelity.

3. **Parsing processes and nodes**: the `parseProcesses()` method iterates over all `process` tags, instantiating a corresponding `Process` object. It then identifies BPMN elements based on a predefined mapping between XML tag names and node classes (`nodeTypeMap`), using `parseNodes()` to instantiate each node with its layout and metadata.

4. **Parsing sequence flows**: within each process, the `parseFlows()` method extracts `sequenceFlow` elements, creating directed connections between previously parsed nodes. These are then linked through the `setNodeArcs()` method, which establishes bidirectional references between source and target nodes for traversal and analysis.

5. **Parsing message flows**: finally, `parseMessageFlows()` collects all inter-process message flows and associates them with the correct source and target nodes. Layouts are also assigned to these flows for consistent rendering.

By the end of the parsing phase, the resulting BPMN object is complete, semantically rich, and layout-aware, serving as the foundation for the subsequent translation into a Petri net.

4.2.1 Converter

The **conversion phase** is the core of the BPMN-to-Petri Net transformation process. The `Converter` module is responsible for interpreting the BPMN diagram and translating it into a semantically equivalent Petri net model. The implementation is encapsulated in the `Converter` class, which takes as input a parsed BPMN object and produces a corresponding `PetriNet` instance.

Upon instantiation, the converter initializes a new Petri net and prepares support structures to manage transitions and arcs that require post-processing, particularly for handling XOR gateways.

The `convert()` method coordinates the entire transformation through the following key stages:

1. **Activity conversion**: each BPMN node representing a task is converted into a Petri net transition. If timed tasks are enabled, the task is split into a start and end transition with an intermediate place.
2. **Gateway conversion**: BPMN gateways are mapped to their equivalent Petri net structures:
    - XOR gateways are handled using custom logic to divide outgoing/incoming arcs into distinct transitions.
    - AND gateways are translated into synchronization structures using ANDSplit/ANDJoin transitions.
    - `Inclusive (OR)` gateways are represented using a hybrid approach involving both XOR and AND combinations.
3. **Flow conversion**: BPMN sequence flows are translated into arcs in the Petri net, preserving their visual structure using layout waypoint data.
4. **Message flow conversion**: inter-pool message flows are processed separately and added to the Petri net while preserving their original semantics.
5. **Initial and final state insertion**: for each BPMN process, an initial and final `place` is added to define entry and exit points in the Petri net.
6. **Global state synchronization**: if multiple pools are present, their initial and final states are unified using an ANDSplit at the start and an ANDJoin at the end, ensuring synchronized execution.
7. **Initial token assignment**: finally, a token is added to the initial place of the net to mark the starting state.

Gateway Management

Each type of BPMN gateway requires specific logic to be correctly mapped to Petri net semantics. The system supports the following:

- **Exclusive Gateway (XOR)**: Models decision points where only one path can be followed. Converted into a `XORSplit` (one place with multiple outgoing transitions) or `XORJoin` (multiple incoming transitions converging to one place). Additional transformations ensure semantic correctness.
- **Parallel Gateway (AND)**: Used for synchronization and concurrency. A split results in one transition activating multiple places simultaneously; a join waits for all inputs before proceeding.
- **Inclusive Gateway (OR)**: Allows conditional execution of one or more paths. Simulated through a combination of XOR and AND transitions that represent all non-empty subsets of the outgoing/incoming flows.

- **Event-Based Gateway**: Waits for external events. Converted into a common place with one transition for each possible event. The first activated transition simulates the event that occurred, preserving the race condition behavior.

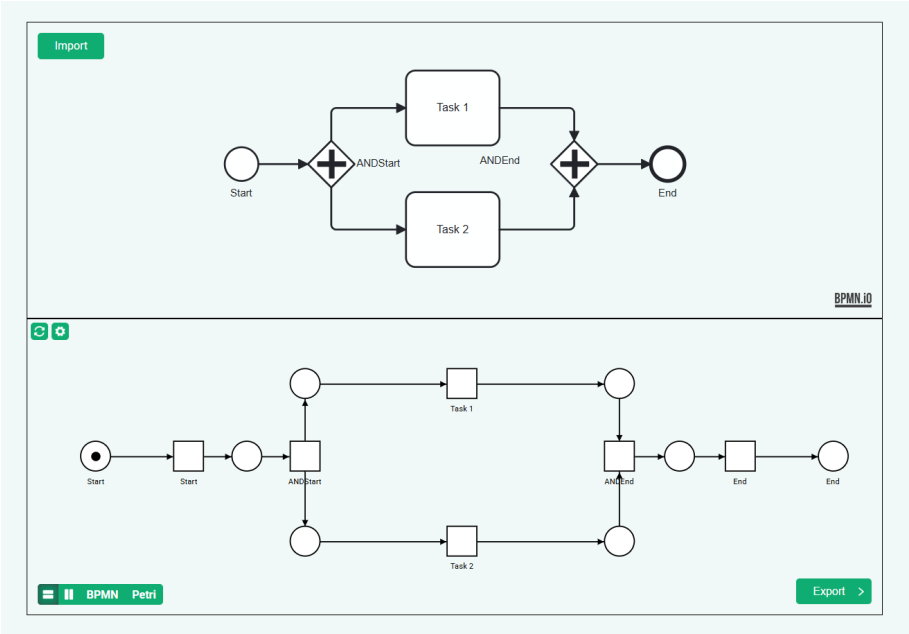

Figure 4.2: Conversion of a parallel gateway

4.2.2 Layout Management

To ensure that the generated Petri net preserves a graphical layout consistent with the original BPMN diagram, the system manages the **layout** using *waypoints*. These act as control points used to trace the arcs of the net, maintaining the visual flow of the original diagram.

Usage of Waypoints and Node Placement

During the conversion from BPMN to Petri Net, each BPMN sequence flow is transformed into an arc, which may optionally include *waypoints*—a sequence of intermediate coordinates used to shape the arc path. These waypoints are stored as an ordered array within each `Arc` object. When a sequence flow directly connects two nodes without deviation, the array remains empty, resulting in a straight arc.

The presence of waypoints not only influences the arc's geometry but also guides the placement of additional nodes (such as intermediate `Place` or `Transition`) introduced to maintain Petri Net semantics. The system adopts the following strategies:

- **With waypoints:** if the BPMN flow includes predefined waypoints, any auxiliary node is positioned directly at the first of these intermediate coordinates.

- **Without waypoints:** in the absence of waypoints, the auxiliary node is placed proportionally along the line connecting the source and target nodes. A fixed percentage (e.g., 30% or 60%) of the segment is used to compute the coordinates.

4.3 Displaying Result

Once the BPMN diagram has been converted into a Petri net, the system provides a graphical representation of the result. This process is managed by the `draw()` method, implemented within the `PetriNet` class and its main components (`Place`, `Transition`, `Arc`).

The `draw()` method of the `PetriNet` class enables direct rendering of the Petri net within the user interface, drawing its components inside a `div`. It iterates over all places, transitions, and arcs, invoking their respective `draw()` functions, assigning their size based on the `NodeSize` setting, and scaling arc lengths according to the `FlowScaling` parameter.

If conversion with decorators is enabled, visual elements are added to the `XORSplit`, `XORJoin`, `ANDSplit`, and `ANDJoin` transitions to clearly indicate their type.

Finally, for each `Place`, if exactly one token is present, a filled black circle is drawn inside it. If more than one token is present, the quantity is displayed as a numeric label within the place.

To make element and arc sizing more flexible, the `draw()` method applies a configurable scaling factor defined by the `displayScale` property.

4.3.1 Layout Rendering with CSS Positioning

The visualization of the Petri net is handled through standard HTML and CSS positioning techniques. All graphical elements—`places`, `transitions`, and `arcs`—are rendered inside a container with `position: relative`, which acts as a reference for positioning internal elements.

Each element is drawn as a `div` with `position: absolute`, using its scaled `x` and `y` coordinates to set the CSS properties `left` and `top`. This allows each element to be placed precisely based on its logical position within the net. The scaling factor is configurable to ensure that the network fits appropriately within the viewport.

Arcs are rendered as rotated horizontal lines between node coordinates, optionally including a directional arrowhead when needed.

4.3.2 Arc Drawing and Transformation

In the rendering phase, arcs between nodes are visualized as horizontal lines with CSS transformations applied to match the actual geometry between the source and target. Each arc is created as an HTML element with a computed width and rotated accordingly.

The length of an arc is calculated using the Euclidean distance between the source point (x_s, y_s) and the target point (x_t, y_t), scaled appropriately:

$$\text{width} = \sqrt{(x_t - x_s)^2 + (y_t - y_s)^2} \cdot \text{scale}$$

To orient the arc correctly, a rotation is applied using the angle θ, which is determined by:

$$\theta = \arctan\left(\frac{y_t - y_s}{x_t - x_s}\right)$$

This angle is passed to the CSS `transform` property using: `transform: rotate(`θ`rad)`

The starting position of the arc is set with `left` and `top`, computed from the source coordinates and adjusted by the node size and scale factor:

$$\texttt{left} = x_s \cdot \text{scale} + \frac{\text{nodeSize}}{2}, \quad \texttt{top} = y_s \cdot \text{scale} + \frac{\text{nodeSize}}{2}$$

Arcs may also include intermediate *waypoints*, which are used to represent multi-segment paths. These are processed as a sequence of connected points: each pair of consecutive points forms a segment drawn with its own rotated line. The final segment connects the last waypoint to the target node.

4.3.3 Zoom Interaction on the Petri Net

The system includes an intuitive zoom mechanism that enhances the exploration of the Petri net model within the interface. This functionality is implemented in the method `onMouseWheel(event)`, which handles user input from the mouse scroll wheel.

When the user scrolls while holding the `Ctrl` key, the method interprets the action as a zoom request. It dynamically adjusts the `scale` CSS transform property of the `.petrinet` container, making the net appear larger or smaller. The zoom factor is calculated proportionally to the scroll delta, ensuring smooth magnification. The current scale is stored in the element's data attributes for consistent behavior.

If the `Ctrl` key is not pressed, the scroll input is instead interpreted as a vertical pan, shifting the network vertically to allow navigation without altering its size. This design improves usability, allowing users to quickly inspect different areas of complex models without losing the overall layout.

4.4 Exporter

The export phase aims to generate a representation of the resulting Petri net in standard formats suitable for further analysis or visualization. The `Exporter` module implements two main export modes: **PNML** and **Graphviz**, providing users with the ability to save and share the generated net.

4.4.1 To .pnml

Exporting in **PNML (Petri Net Markup Language)** format allows the system to produce an XML-based representation of the Petri net, compatible with various analysis and simulation tools. The main method responsible for this operation is `export()`, which generates the corresponding XML document.

PNML File Structure

The exported file follows the standard PNML structure and includes the following elements:

- **Net**: the root element representing the Petri net.
- **Places**: the places of the Petri net, including position and initial token count.
- **Transitions**: the transitions of the Petri net, with associated coordinates and labels.
- **Arcs**: the arcs connecting the nodes, optionally including *waypoints* to preserve the original layout.

Export Process

The export is carried out through the following steps:

1. Creation of an XML document with a `pnml` root node.
2. Creation of the `net` node, which will contain all elements of the Petri net.
3. Iteration over the Petri net nodes to export places, transitions, and arcs, including optional decorations for AND and XOR gateways.
4. Generation of the final XML and formatting using the `prettifyXml()` method. This function takes the raw XML string and reprocesses it using an XSLT stylesheet to produce an indented and human-readable version.

4.4.2 To .dot

Exporting in **Graphviz (`.dot`)** format allows the system to produce a textual representation of the Petri net without the need to define the exact position of each element. Instead, the layout is automatically handled by visualization tools, which apply specific algorithms to ensure the diagram remains readable. The `exportGraphviz()` method generates a file compatible with Graphviz, enabling clear and elegant visualization of the net.

DOT File Structure

The generated file follows the DOT syntax and includes:

- A `digraph G` declaration that defines the Petri net as a directed graph.
- A series of nodes representing places and transitions, with custom attributes:
  - Places are rendered using the `circle` shape.
  - Transitions are rendered using the `square` shape.
- A series of directed edges connecting the nodes.

Export Process

The export to `.dot` format follows these steps:

Algorithm 2 Exporting the Petri net to DOT format

```
dotString ← "digraph G {"
for node in petrinet.nodes do
    dotString ← dotString + formatNode(node)
end for
for arc in petrinet.arcs do
    dotString ← dotString + formatArc(arc)
end for
dotString ← dotString + "}"
return dotString
```

Customization Options

The user can choose whether to place node labels inside or outside their corresponding shapes using the `textOutside` parameter. Specifically:

- If `textOutside` is enabled, the node name is added as an `xlabel`, displayed outside the shape.
- If `textOutside` is disabled, the name is included directly inside the node shape using the `label` attribute.

Chapter 5

Additional Features

Beyond the standard conversion of BPMN diagrams into Petri nets, the system implements a series of additional features that improve flexibility and the quality of the generated result. These features include advanced methods for handling complex elements, enhancements to the readability of the resulting model, and customization options to adapt the Petri net to the user's specific needs. The main implemented extensions are described below.

5.1 Configuration Panel

To enhance flexibility and adaptability, the interface includes a dedicated **configuration panel** that allows users to customize the behavior of the BPMN-to-Petri Net conversion process. This panel, shown in Figure 5.1, is accessible via a collapsible menu located in the user interface.

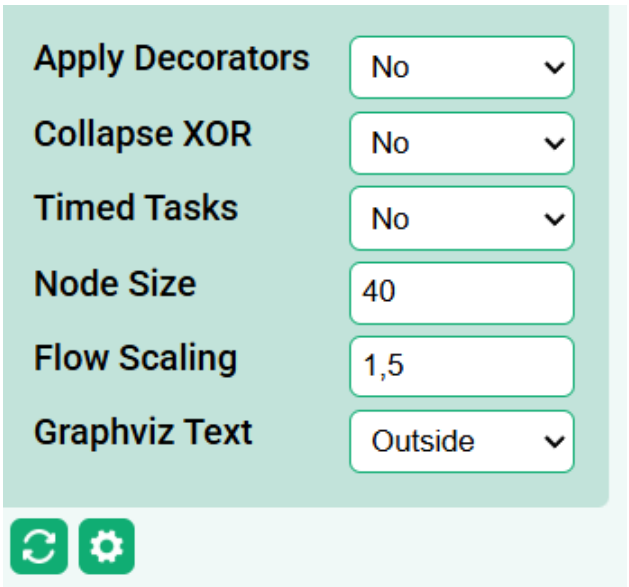

Figure 5.1: Configuration menu for customizing the conversion process

Each configuration option is bound to a global `Config` object that directly influences how the Petri Net is generated and rendered. Once parameters are changed, clicking the **refresh icon** ( ) reloads the diagram with the updated settings, applying them to the visualization and conversion process. The panel can be shown or hidden using the **toggle icon** ( ).

The configurable parameters are:

- **Apply Decorators**: enables visual indicators on transitions (AND/XOR `split/join`) to clarify control-flow semantics without altering the net structure.
- **Collapse XOR**: enables a simplified representation of XOR gateways by removing intermediate transitions and directly connecting places to task transitions when logic allows.
- **Timed Tasks**: when enabled, each task is split into a start and end transition with an intermediate place, enabling temporal analysis and simulation.
- **Node Size**: defines the rendered size (in pixels) of each place and transition node, improving readability on different displays.
- **Flow Scaling**: adjusts the length of arcs between nodes. Larger values stretch the arcs for better clarity in complex models.
- **Graphviz Text**: sets whether labels in Graphviz (`.dot`) exports are placed `inside` or `outside` the nodes.

5.2 OR Conversion

The conversion of **OR gateways** from BPMN to Petri nets is one of the most challenging transformations due to the dynamic nature of the choices this element introduces. An OR gateway allows one or more outgoing paths to be activated depending on certain conditions, which makes its direct representation in Petri nets more complex.

Using OR gateways in Petri nets provides greater expressiveness, allowing the modeling of scenarios where multiple paths may be conditionally executed, without the rigidity imposed by XOR or AND.

However, using OR gateways may compromise key Petri net properties, such as **soundness** and **boundedness**. In particular, the network can become *unbounded* when an OR split activates multiple paths and the corresponding join synchronizes only a subset of them. Unsynchronized tokens continue to propagate indefinitely, generating an unbounded state space.

To manage all possible path combinations, the OR gateway is converted into a series of AND Splits and AND Joins. If the OR gateway has n outgoing paths, the number of AND transitions required is $2^n - 1$, leading to exponential growth that impacts readability and efficiency.

Algorithm 3 Conversion of an Inclusive Gateway to a Petri Net

```
1:  procedure convertInclusiveGateway(gateway, process)
2:      coords ← getCoordinates(gateway)
3:      if gateway.type == OR-Split then
4:          split ← createPlace(gateway, process)
5:          places ← ∅
6:          for arc ∈ gateway.outgoingArcs do
7:              transition ← createTransition(arc.target, process)
8:              place ← createPlace(arc.target, process)
9:              split → transition,   transition → place,   place → arc.target
10:             places ← places ∪{place}
11:         end for
12:         for subset ∈ powerset(places), |subset| > 1 do
13:             andSplit ← createAndSplit(gateway, process)
14:             split → andSplit,   ∀ place ∈ subset: andSplit → place
15:         end for
16:     else if gateway.type == OR-Join then
17:         join ← createPlace(gateway, process)
18:         places ← ∅
19:         for arc ∈ gateway.ingoingArcs do
20:             place ← createPlace(arc.source, process)
21:             transition ← createTransition(arc.source, process)
22:             arc.source → place,   place → transition,   transition → join
23:             places ← places ∪{place}
24:         end for
25:         for subset ∈ powerset(places), |subset| > 1 do
26:             andJoin ← createAndJoin(gateway, process)
27:             ∀ place ∈ subset: place → andJoin,   andJoin → join
28:         end for
29:     end if
30: end procedure
```

Legend: the → operator denotes the creation of an arc between nodes.

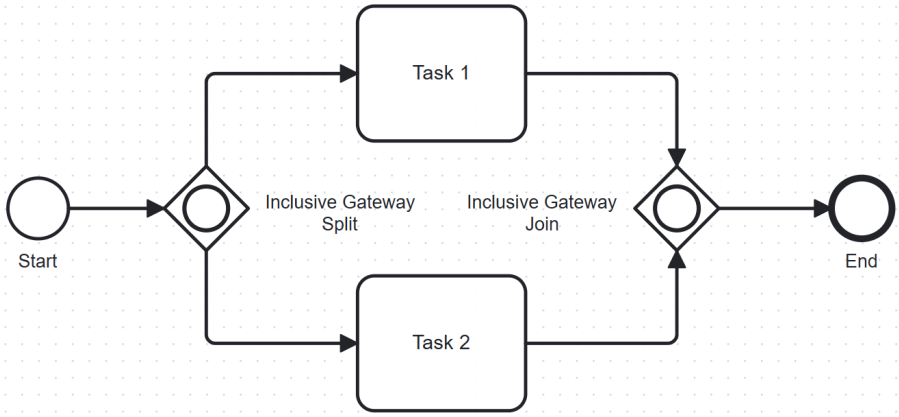

Figure 5.2: Inclusive gateway before conversion

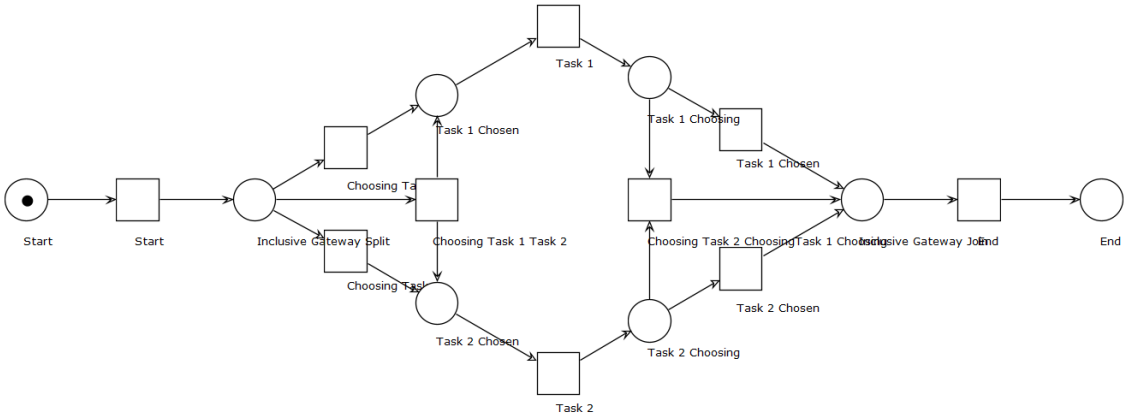

Figure 5.3: Inclusive gateway after conversion

5.3 Decorators

Decorators enhance the readability and understanding of the generated Petri net. Specifically, they clarify logical gateways and improve structural analysis without altering net behavior.

During the conversion of BPMN elements such as XOR and AND gateways, the resulting Petri net may become difficult to interpret, especially with many connections. To address this, graphical decorators are used to preserve the visual representation of gateway semantics.

The framework implements two main types of decorators. XOR decorators visually differentiate between *XOR-Split* and *XOR-Join*, while AND decorators help distinguish between *AND-Split* and *AND-Join*, even though their behavior is equivalent to standard transitions.

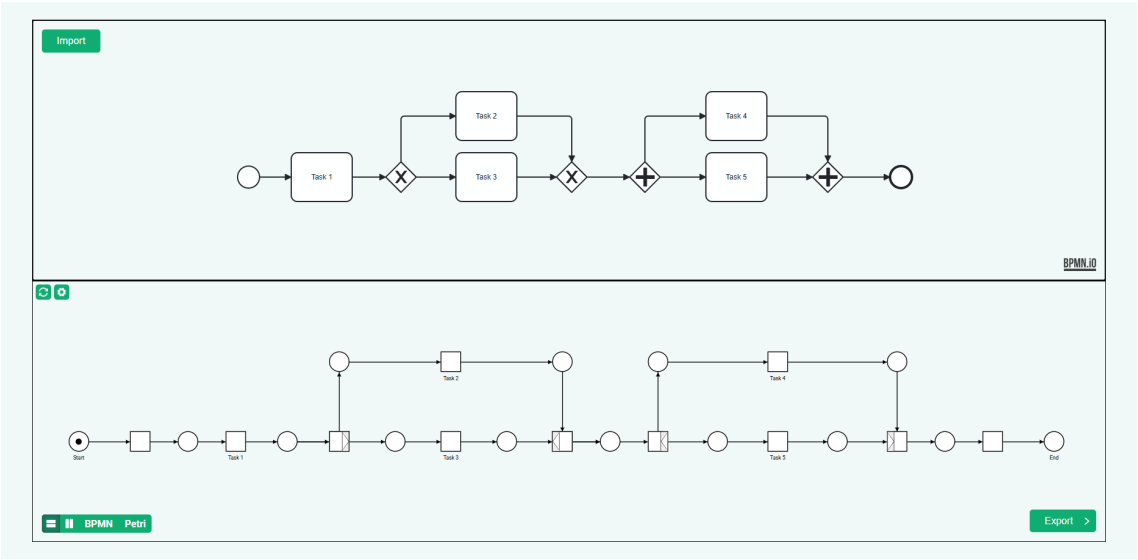

Figure 5.4: Conversion with the decorators setting on

5.4 Collapsed XOR

In the default conversion, *XOR-Splits* are represented as a **place** followed by multiple **transitions**, and *XOR-Joins* use intermediate transitions before merging into a single place. This ensures correctness but may introduce structural complexity.

To reduce the number of nodes, an optimization called **Collapsed XOR** was introduced. When enabled, the intermediate transitions are skipped, and places connect directly to task transitions when possible, simplifying the structure while preserving semantics.

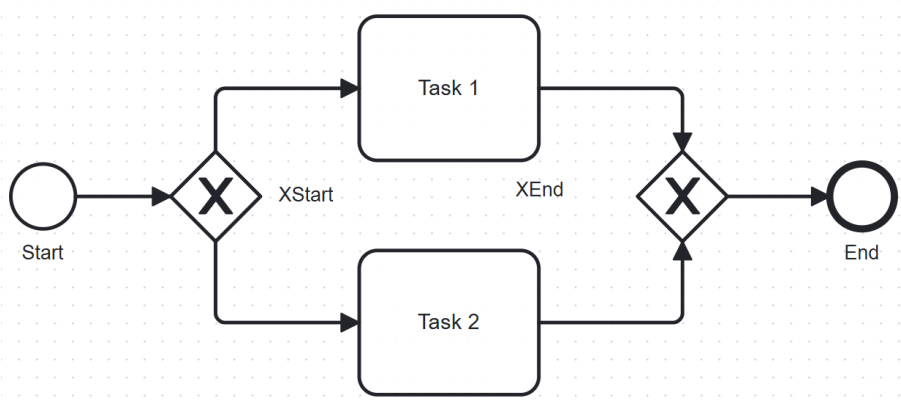

Figure 5.5: Exclusive gateway before conversion

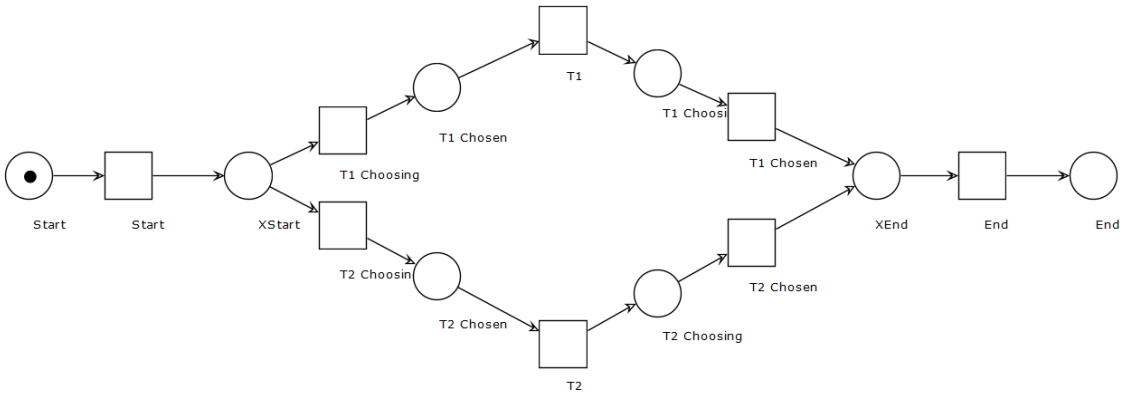

Figure 5.6: Exclusive gateway after conversion with intermediate nodes

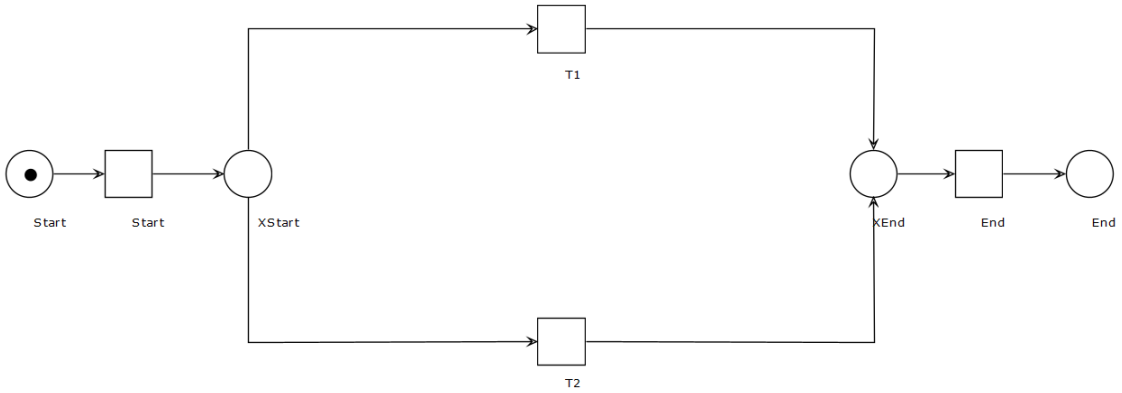

Figure 5.7: Exclusive gateway after conversion with collapsed intermediate nodes

5.5 Timed Tasks

To better represent execution duration, the system allows tasks to be modeled as **Timed Tasks**, explicitly dividing each into *start* and *end* phases.

Each BPMN task is split into two transitions:

- **Start Task**: represents the activation of the task, enabled once all predecessors are complete.
- **End Task**: represents the task's completion, enabled only after the Start Task has fired.

An intermediate **place** is inserted between them to hold the token during execution. This structure enables modeling of wait conditions, timeouts, or delays.

Timed Tasks improve simulation accuracy by separating activation from completion and enable deeper time-based analysis.

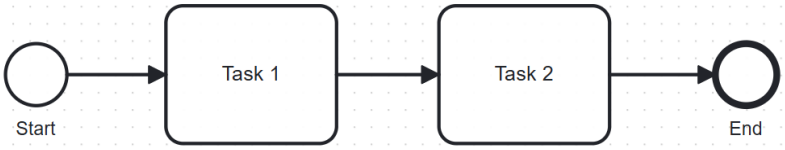

Figure 5.8: Simple sequential BPMN

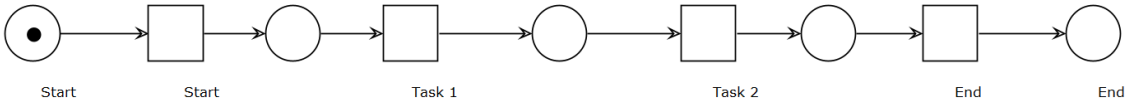

Figure 5.9: Simple sequential BPMN converted without timed tasks

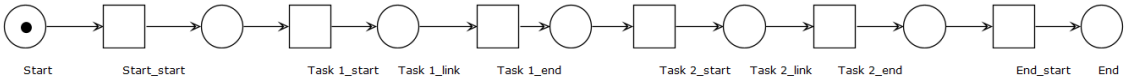

Figure 5.10: Simple sequential BPMN converted with timed tasks setting on

5.6 Batch Mode

To improve the efficiency of converting multiple BPMN diagrams, the system includes a **Batch Mode** feature, allowing multiple files to be uploaded and processed simultaneously.

In the homepage, users can upload files either through a file input button or by dragging them into the designated area (*drag & drop* support).

Once uploaded, diagrams are converted immediately, and the following actions are available for each:

- **Individual visualization**: each generated Petri net can be opened in a new tab via the open icon ( ), using the browser's `localStorage` to transfer BPMN content.

- **Individual export**: each converted file can be downloaded using the download icon ( ).

A collective export button is also available to download all generated Petri nets with a single click, providing an efficient output management workflow.

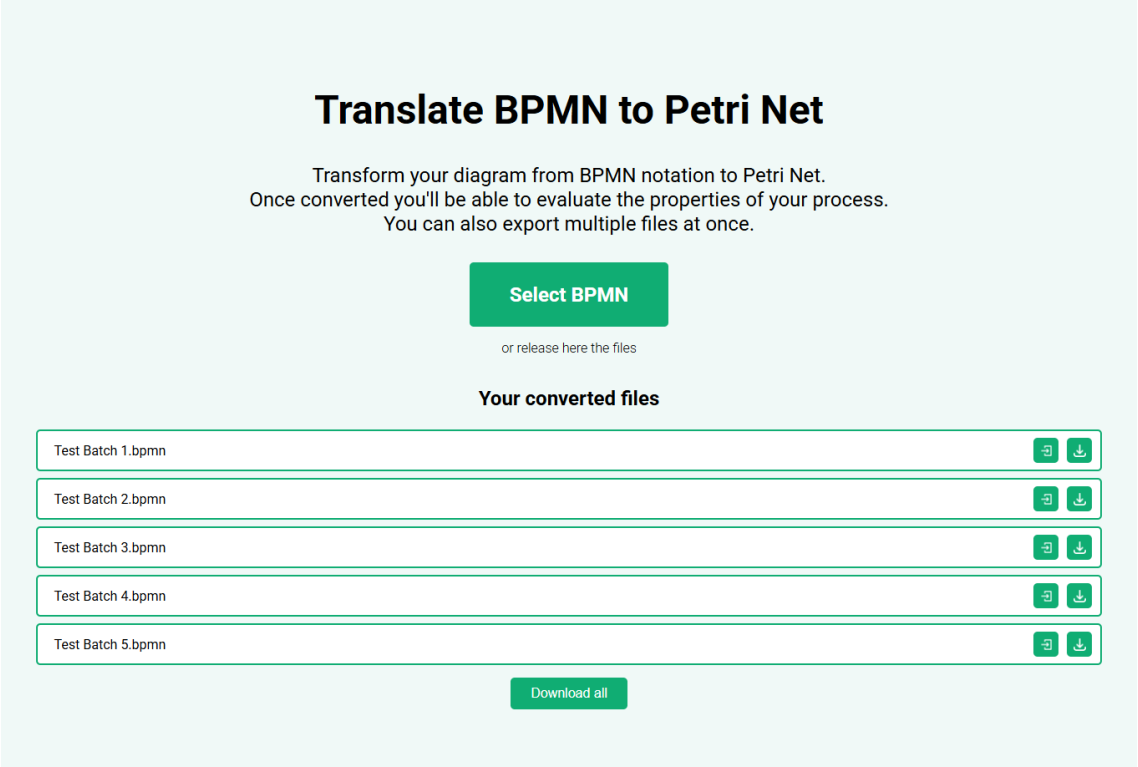

Figure 5.11: Homepage after submitting multiple BPMN files

Chapter 6

Results & Testing

To validate the functionality and correctness of the BPMN to Petri Net conversion system, we conducted a series of tests using various BPMN diagrams. Each test case targets a specific BPMN structure or behavior, and the resulting Petri Net is analyzed for correctness, structural coherence, and semantic preservation.

This chapter presents the visual results of the conversion process, highlighting how different BPMN elements are transformed into their Petri Net counterparts. Screenshots from the application interface are provided to illustrate both the original BPMN diagram and the generated Petri Net.

6.1 Arcs Waypoints

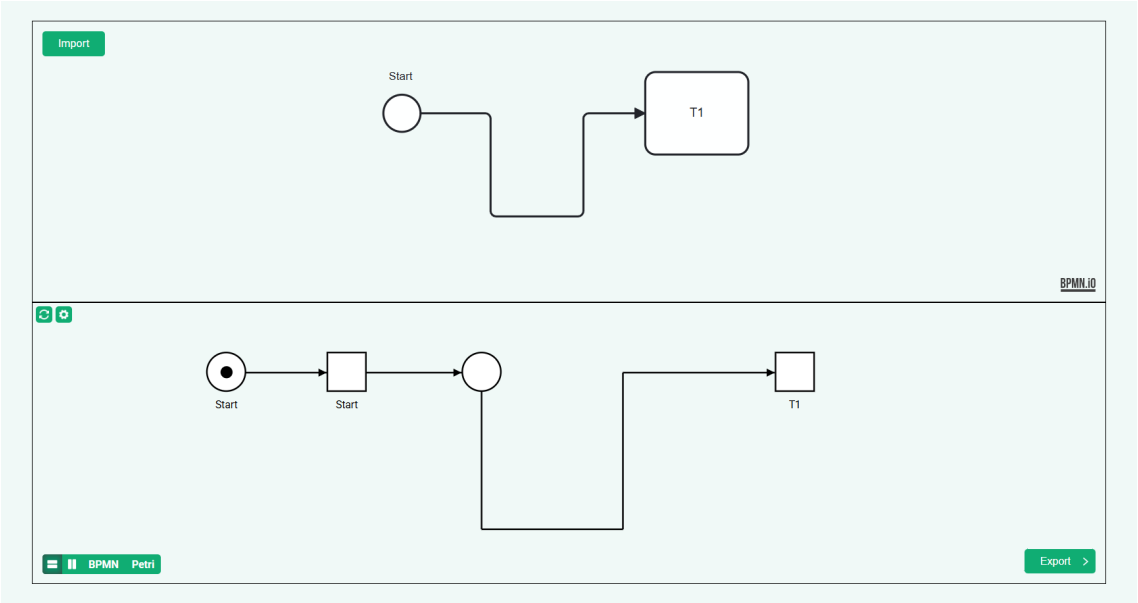

Figure 6.1: Conversion of a diagram with curved connectors

In this test, a simple sequential BPMN process with curved connections was analyzed. The conversion preserved the visual layout of the original diagram thanks to the use of *waypoints* defined in the BPMN file. Specifically, during the transformation, the additional `place` introduced to comply with Petri net semantics was positioned exactly at the point corresponding to the first waypoint of the BPMN flow.

6.2 Inclusive Gateway (OR)

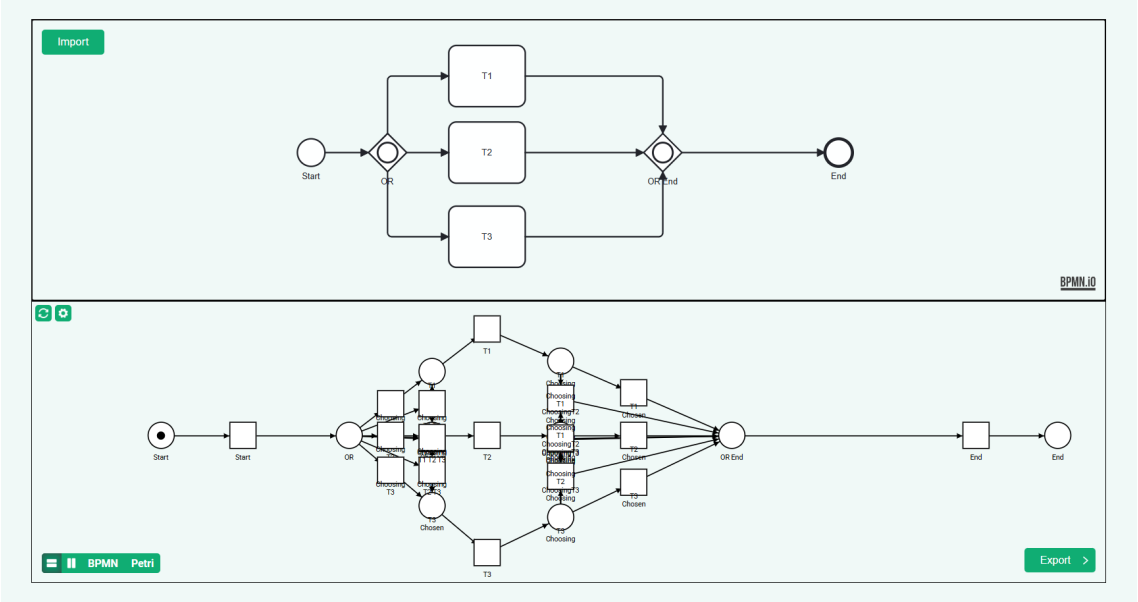

Figure 6.2: Conversion of an Inclusive Gateway with three outgoing tasks

This test case was designed to evaluate the conversion of an **inclusive gateway (OR)** with three outgoing tasks. The goal was to observe the exponential growth in the number of nodes required to represent all possible combinations of path activations. As expected, the resulting Petri Net includes multiple AND-splits to account for every non-empty subset of outgoing branches, leading to a significant increase in structural complexity. Consequently, the resulting Petri Net is notably less readable, highlighting one of the main trade-offs in supporting OR gateways with full semantic fidelity.

6.3 Simple Message Flow Between Pools

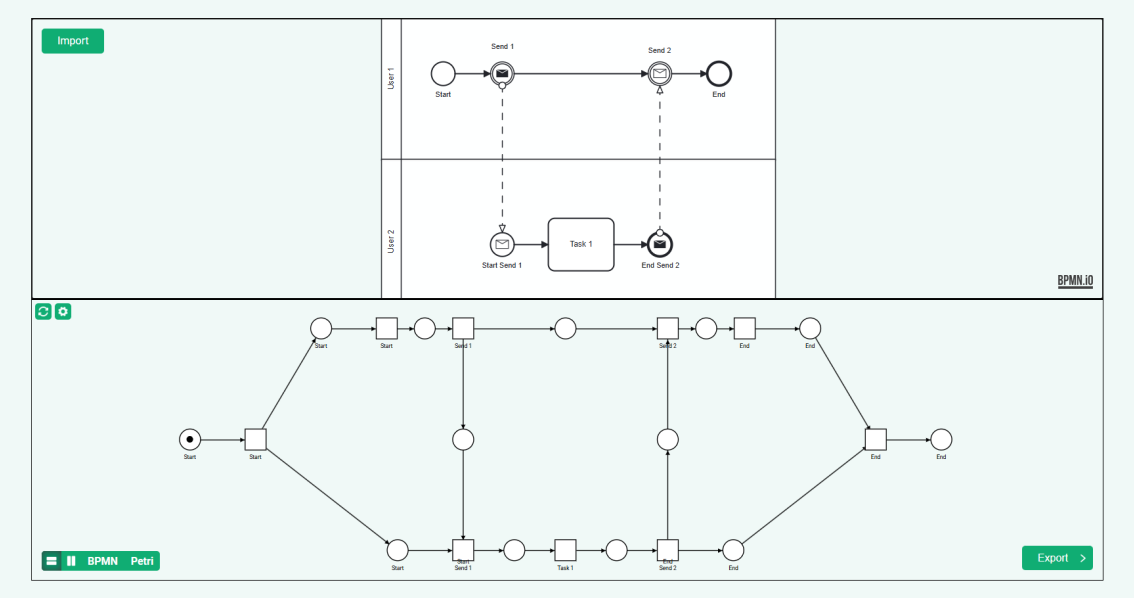

Figure 6.3: Basic message flow between two users

In this test, we have modeled the exchange of messages between two distinct processes. In the resulting Petri net, we observe that **catch message events** (i.e., events that receive a message) are represented by transitions with two incoming arcs: one from the regular control flow and another from the message flow. This ensures that the transition can only fire once the process reaches the event and the message is received from the other process.

Conversely, **throw message events** (i.e., events that send a message) are represented by transitions with two outgoing arcs: one continuing the regular control flow, and one targeting the receiving process through a message arc. This structure guarantees that sending a message both advances the local process and triggers the corresponding receive event on the other side.

Finally, the two processes are merged into a single Petri net by adding shared final places and transitions, ensuring proper synchronization and completion of all flows.

6.4 Event-Based Gateway

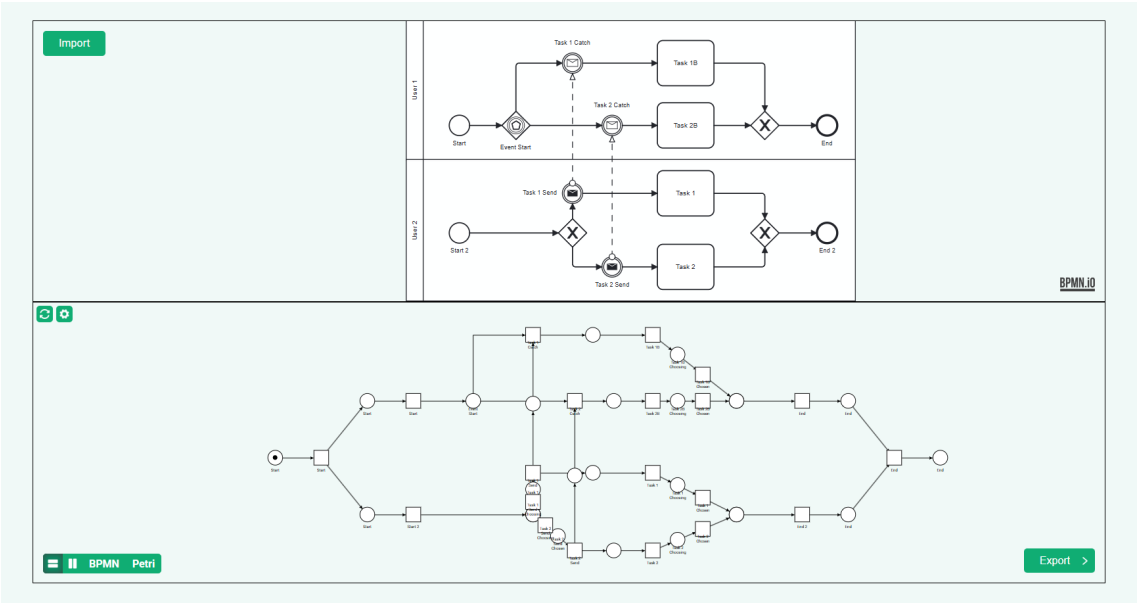

Figure 6.4: Conversion of an event-based gateway and message catching

In this test, we present another scenario involving two parallel processes. This time, however, the focus is on evaluating the behavior of the **event-based gateway**, which was assigned to *User 1*'s process. The gateway waits for one of two incoming message events—both of which are triggered by *User 2*, who sends messages via distinct flows.

The generated Petri net may appear more intricate due to the increased number of transitions and synchronization points, yet it correctly reflects the intended semantics: *User 1* will proceed based on the first message received from *User 2*, effectively capturing the exclusive and event-driven nature of the original BPMN model.

6.5 Message Flow Starting from Task

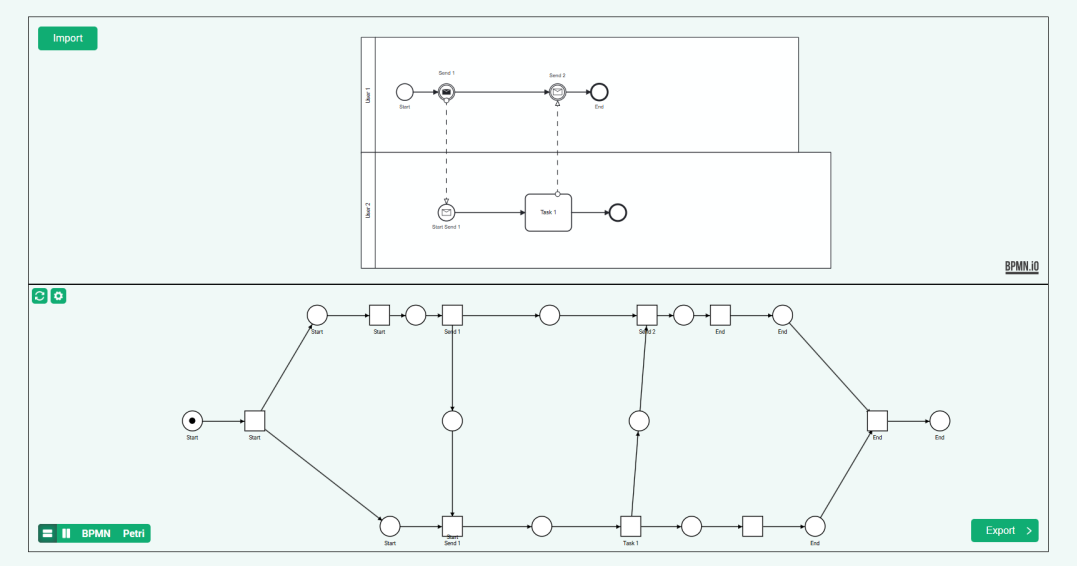

Figure 6.5: Message flow sent directly from a task

This test demonstrates that message flows can also be initiated directly from a task, without relying on a separate throw event. As shown in Figure 6.5, the sending action is embedded in the task itself, and the conversion correctly produces a Petri net where the corresponding transition emits a token through both the process flow and the message arc. This confirms that message sending is supported not only via explicit *throw events*, but also as an integrated behavior within tasks.

6.6 Visualization Modes

To facilitate the comparison between BPMN diagrams and their corresponding Petri net representations, the system provides multiple visualization modes. These modes are particularly relevant when analyzing models developed with a strong vertical structure, such as those involving sequential flows or stacked gateways.

Among these options, the side-by-side view (Figure 6.6) is the most effective when dealing with diagrams that follow a top-down development pattern. This layout places the BPMN diagram on the left half and the corresponding Petri net on the right half of the interface, enabling a direct horizontal comparison of the same segments in both notations. In contrast, the horizontal split layout (Figure 6.7) may require excessive scrolling or significant zooming out when dealing with vertically extended diagrams, which can hinder the clarity and immediacy of the comparison.

The system also allows single-view modes (Figures 6.8 and 6.9), which are useful during focused inspection or when exporting one representation at a time. These modes remove the additional context but can improve readability in dense or detailed models.

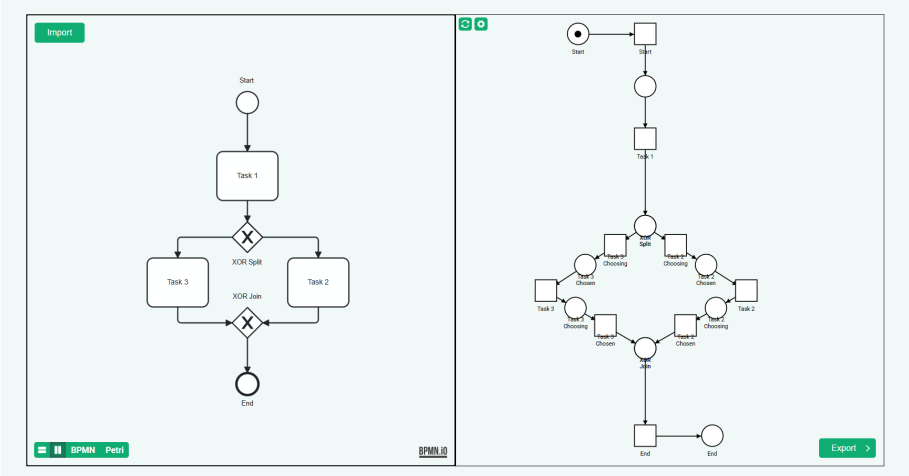

Figure 6.6: Vertical layout with horizontal split: BPMN above, Petri net below

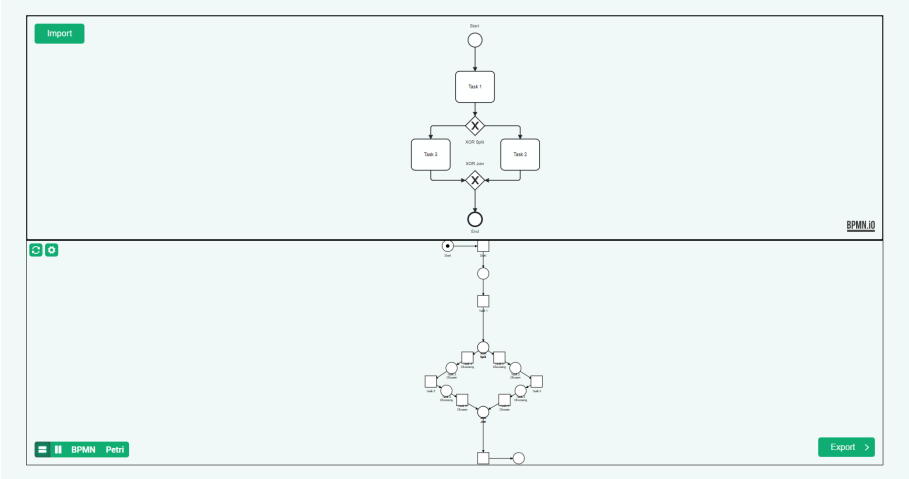

Figure 6.7: Horizontal layout with side-by-side BPMN and Petri net

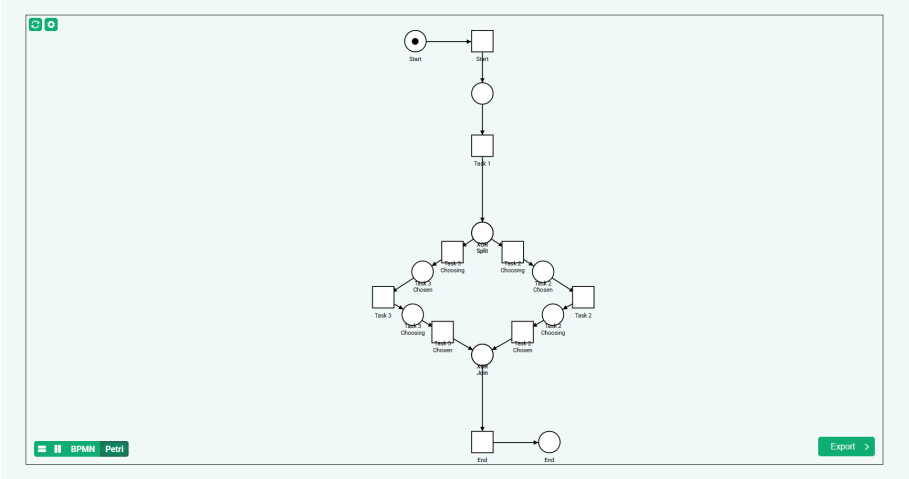

Figure 6.8: Single view: only the Petri net is shown

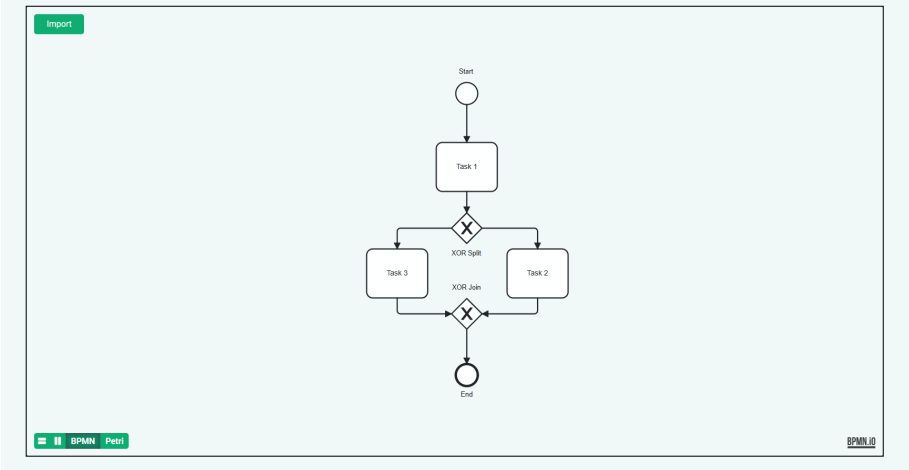

Figure 6.9: Single view: only the BPMN diagram is shown

6.7 Export Functionality

The system allows exporting the resulting Petri Net in both `.pnml` and `.dot` formats. While the `.pnml` format ensures compatibility with external analysis tools, the `.dot` format is particularly useful for graph visualization through layout-optimized renderers.

For the DOT export, two rendering modes are available:

- **Internal labels**: labels are rendered within the node shapes (Figure 6.13), using the standard `label` attribute.
- **External labels**: labels are positioned outside the node shapes to enhance readability (Figure 6.12), by assigning text to the `xlabel` attribute and setting the `label` attribute to an empty string.

The generated DOT code can be visualized using dedicated tools that automatically optimize the layout. For testing purposes, the online tool `https://dreampuf.github.io/GraphvizOnline/` was used, which offers immediate rendering from DOT code. The Petri net representation was visualized using **WoPeD**, a dedicated tool for modeling and analyzing Petri nets.

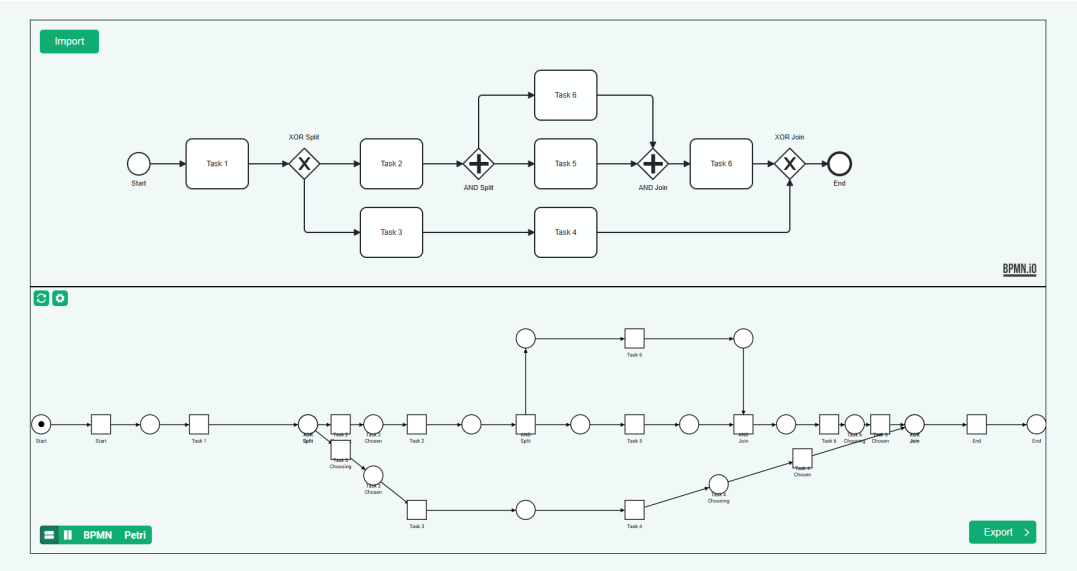

Figure 6.10: BPMN diagram and corresponding Petri Net in the main interface

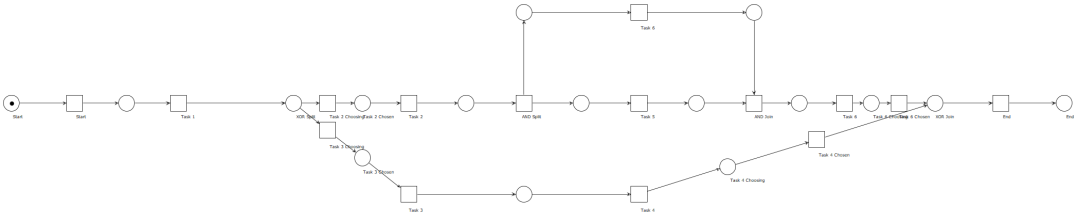

Figure 6.11: .pnml export visualized in WoPed

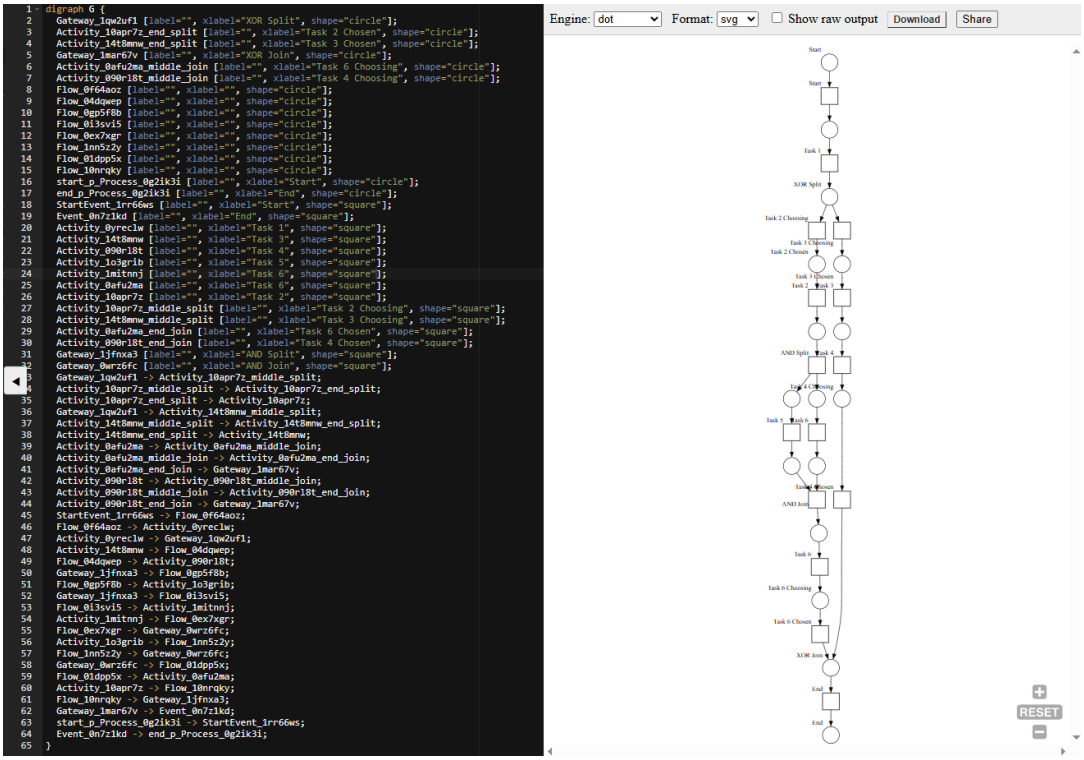

Figure 6.12: DOT export with external labels

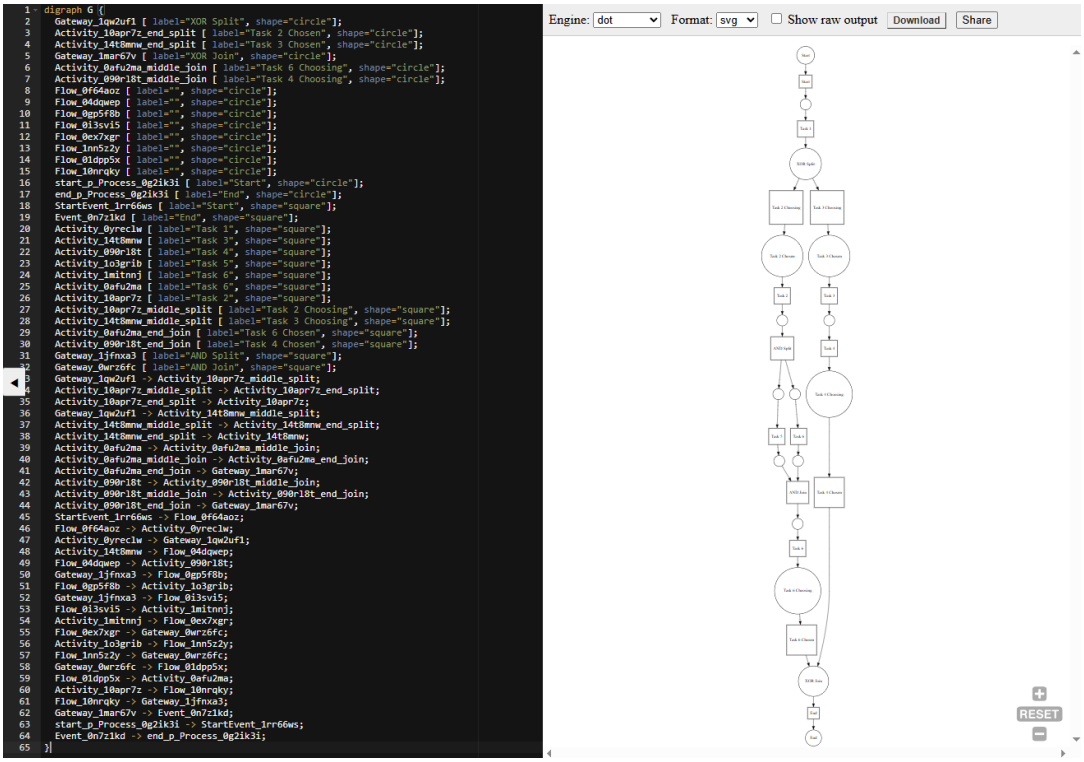

Figure 6.13: DOT export with internal labels

6.8 Exporting Individual Pools

As a continuation of the test already discussed in Figure 6.4, which involved the use of an event-based gateway and inter-process message flows, we now present the exported Petri nets for each BPMN pool separately.

This export mode removes the *message flows*, isolating the behavior of each participant while preserving the semantics of the overall process. Each Petri net retains its internal structure, ensuring that the logic of message reception and synchronization remains correctly represented, but only from the perspective of the individual process. This format is particularly useful for modular analysis or per-user simulation. The resulting PNML files were visualized using WoPeD.

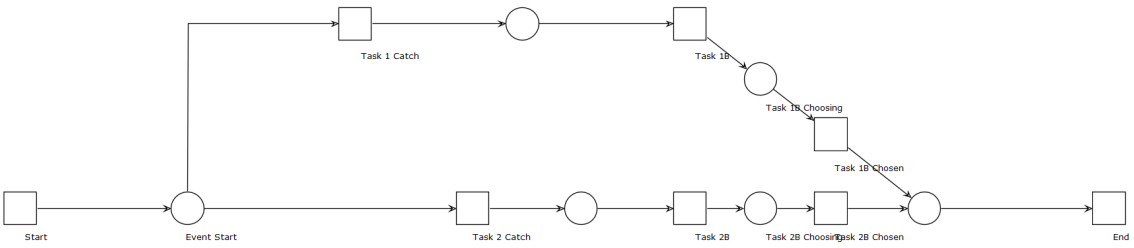

Figure 6.14: Petri net of User 1 exported from the test shown in Figure 6.4.

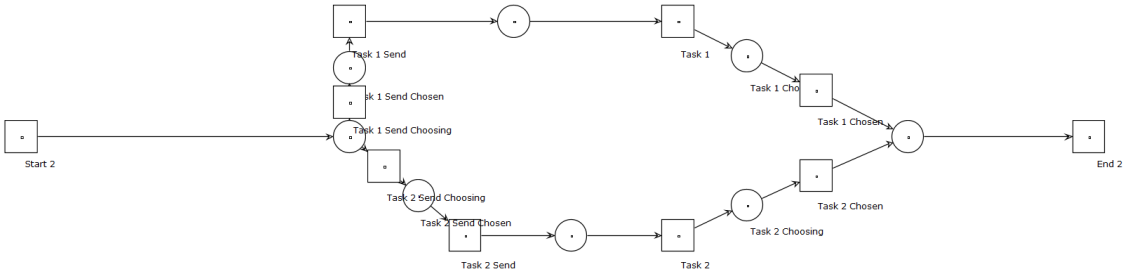

Figure 6.15: Petri net of User 2 exported from the test shown in Figure 6.4.

6.9 Complex BPMN

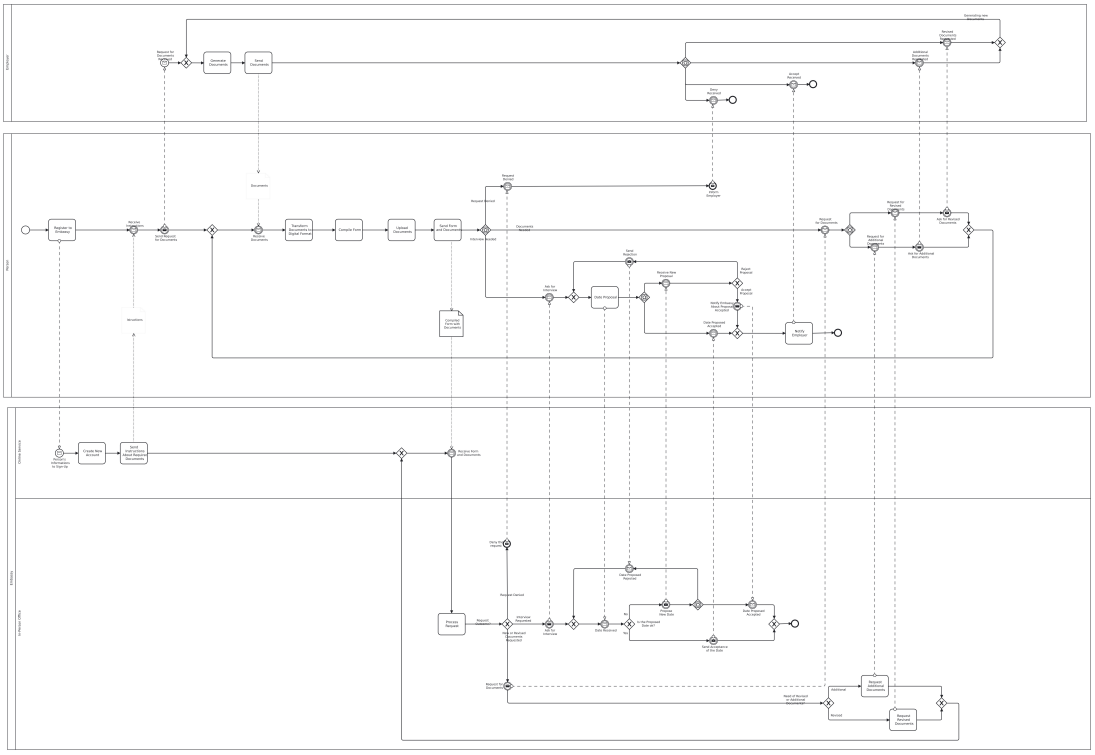

Figure 6.16: BPMN Diagram for the Visa Application Process.

The scenario represented in Figure 6.16 involves a detailed process for visa application, including several decision points, such as the possibility of visa denial, additional document requests, or the scheduling of an interview.

Despite the complexity of this scenario, the developed system was able to consistently translate the BPMN diagram into a coherent Petri net (Figure 6.17).

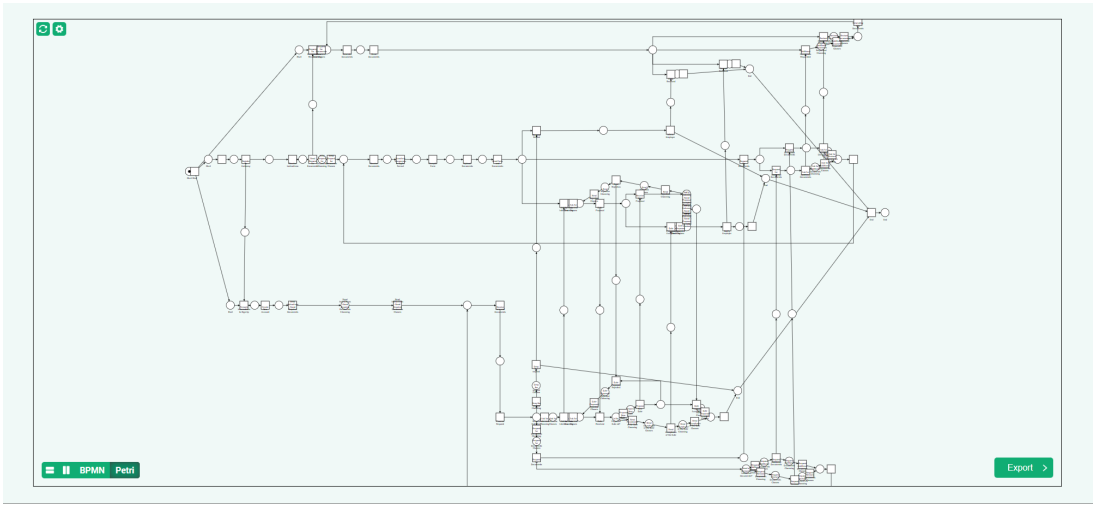

Figure 6.17: Petri Net Translation of the Visa Application Process.

Chapter 7

Conclusions

The developed system offers a robust and extensible framework for the automated translation of BPMN diagrams into Petri Nets, combining usability, formal rigor, and customization. By preserving both the structural and behavioral semantics of the original process models, the tool ensures that the resulting Petri nets are not only visually faithful but also suitable for formal analysis. The modular architecture supports a wide range of BPMN elements and introduces advanced features such as decorators for enhancing semantic detail, message flow management, and dedicated support for complex constructs including inclusive, exclusive, and parallel gateways.

The converter also includes multiple rendering modes, export functionalities (e.g., PNML and DOT formats), and a batch processing mechanism, enabling it to serve diverse use cases—from educational settings and teaching environments to in-depth analytical scenarios involving large-scale process models. The test cases conducted confirm the tool's correctness and adaptability across different diagram types, highlighting the importance of both graphical clarity and semantic precision during the transformation.

Future Work. Several improvements and extensions can further enhance the capabilities of the system:

- **In-place BPMN editing:** Integrating direct editing capabilities within the web interface would allow users to modify BPMN diagrams on the fly and observe real-time updates in the corresponding Petri nets. This would support an interactive modeling experience, facilitating experimentation and incremental refinement.

- **Manual adjustment of Petri Net layout:** Allowing users to move and rearrange Petri net elements after generation would significantly improve the clarity of the resulting graphs, particularly for complex models. This feature would also aid in educational and presentation contexts, where visual readability is crucial.

- **Integrated formal analysis:** Embedding native support for Petri net property analysis—such as reachability, soundness, liveness, and deadlock detection—would eliminate the dependency on external tools like WoPeD or Woflan. This would streamline the validation process and provide immediate feedback on model correctness.
- **Advanced handling of inclusive gateways:** The current approach to OR gateways compromise the soundness of the resulting net. Future improvements could involve implementing more advanced translation strategies to ensure semantic completeness without sacrificing formal properties.

Ultimately, this work demonstrates that it is possible to reconcile the intuitive nature of BPMN with the formal rigor of Petri nets within a modern and accessible technological framework. Rather than forcing users to choose between readability and analyzability, the proposed system unifies both dimensions, offering a concrete tool that supports model refinement, validation, and exploration. By enabling seamless transitions between design and verification, the platform not only addresses current limitations, but establishes a crucial link between intuitive modeling and formal validation. This integration empowers users to move fluidly from conceptualization to rigorous analysis without leaving the same environment. As such, it lays the groundwork for a new generation of modeling tools—ones that no longer treat usability and formalism as opposing forces, but as complementary dimensions of the same process-driven vision.

Bibliography

[1] Shoopi. *BPMN 2 Petri Nets Transformation*. https://github.com/shoopi/petrinet-bpmn-transformation. Accessed: 2025-03-07. 2015.

[2] M. Fahland, B. F. van Dongen, and W. M. P. van der Aalst. "Discovering, Analyzing and Enhancing BPMN Models Using ProM". In: *CEUR Workshop Proceedings*. Vol. 1295. 2014. URL: https://ceur-ws.org/Vol-1295/paper21.pdf.

[3] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. 3rd. Springer, 2019. URL: http://bpm-book.com.

[4] Marlon Dumas et al. *Fundamentals of Business Process Management*. 2nd ed. Springer, 2018. ISBN: 978-3-662-56509-4. URL: http://fundamentals-of-bpm.org/.

[5] Wil M. P. van der Aalst and Kees M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002. ISBN: 0-262-01189-1.

[6] Object Management Group. *Business Process Model and Notation (BPMN) Version 2.0*. 2010. URL: https://www.omg.org/spec/BPMN/2.0/.

[7] Matthias Kunze and Mathias Weske. *Behavioural Models - From Modelling Finite Automata to Analysing Business Processes*. Springer, 2016. ISBN: 978-3-319-44958-6. DOI: 10.1007/978-3-319-44960-9.

[8] Carl Adam Petri. "Kommunikation mit Automaten". PhD thesis. Institut für Instrumentelle Mathematik, Bonn, 1962.

[9] Wolfgang Reisig. *Petri Nets, An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer, 1985.

[10] Wolfgang Reisig. *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013. ISBN: 978-3-642-33277-7. DOI: 10.1007/978-3-642-33278-4.

[11] Wolfgang Reisig. *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer Publishing Company, Incorporated, 2013. ISBN: 3642332773.

[12] Wil M. P. van der Aalst and Christian Stahl. *Modeling Business Processes - A Petri Net-Oriented Approach*. MIT Press, 2011. ISBN: 978-0-262-01538-7.