System Specification

Extended Petri net Simulator

Group A

October 25, 2013

Abstract

This document includes the requirements for the Software Engineering 2 project, which consists of an Extended Petri net software system.

Contents

1 Introduction

Author: ***Thibaud***

> Petri nets, as graphical and mathematical tools, provide a uniform environment for modelling, formal analysis, and design of discrete event systems. [1]

Petri nets are used as a means to model systems, but, as they are a mathematical concept, they are not always easy to understand for the ordinary user. Complex systems can be modelled with Petri nets, and usually, this would be an engineer's job. Even though engineers can easily create and understand their own Petri nets, every team member in a company would like to be able to understand what a Petri net is about without having any knowledge of the concepts behind them.

Therefore, the project aims at creating a 3D visualisation from a Petri net, to allow non-Petri net experts to actually to understand how the model works and validate a system.

However, Petri nets were not intended to have a 3D representation. For instance, there is no graphical concept or way to say that a particular Petri net would look like a train track because it was used to model a railway system.

Thus, our goal for this project is the following: Providing an extension to Petri net models to make their 3D visualisation possible. For this purpose, we have imagined a simple link between a Petri net and a 3D visualisation.

2 Overall description

This section describes the software and provides a brief explanation on how the system works.

2.1 What are Petri nets?

Petri nets are a graphical and mathematical modelling tool for describing concurrent and distributed systems. Some examples of their applications are workflow management, embedded systems or traffic control. The main advantages of Petri nets are their graphical notation, their simplicity on the semantics, and their rich theory for analysing their behaviour. However, using the Petri net graphical notation for understanding a complex system is quite hard, and thus a user-oriented visualization is required in a way that is understandable to users which are not necessarily familiar with Petri nets.

Figure 1 presents an example of a basic Petri net. There are four different elements:

- Places: graphically represented by circles, represent conditions.
- Transitions: graphically represented by squares, represent events.
- Arcs: indicate which places are preconditions/postconditions for which transitions.

[1] Petri Nets and Industrial Applications: A Tutorial. Richard Zurawski, MengChu Zhou.

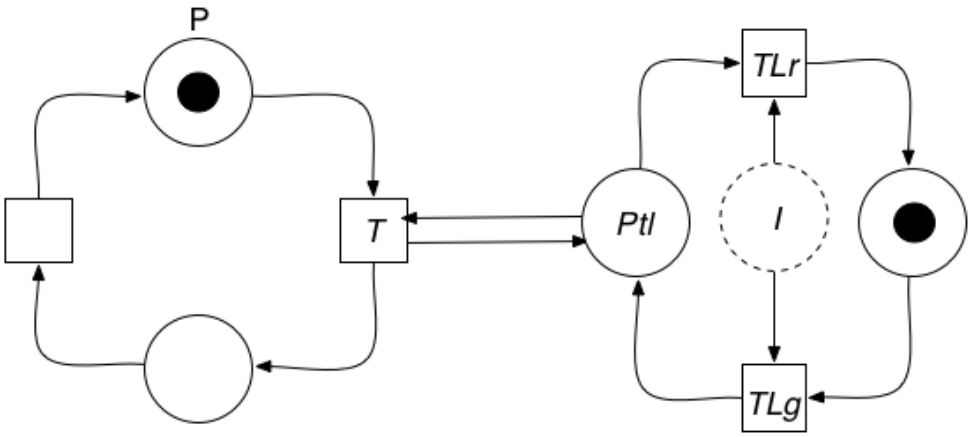

Figure 1: An example of a basic Petri net

- Tokens: graphically represented by the dots, represent the elements that move in the Petri net along the places through the transitions.

Furthermore, we can see the example of Figure 1 as a model for a railway with a traffic light. The token in Place P1 represents a train moving on a railway. When this train arrives to Place P1, the transition T will fire if the conditions are met. The only condition for a transition to be fired is that a token should be present on each of the incoming places of the transition.

In this scenario, the conditions are not met, as the required token in Place Ptl is missing. The visualization of this scenario would be a traffic light with a red light. If Place I had a token on it, it would generate a token that will turn the traffic light to green and thus the train will move.

With the aim of creating a visualization of the scenarios for the above Petri net model, an application tool is needed in order to allow the user to define where each of the elements are represented in the 3D world (geometry editor) as well as how these elements are represented (shape) and how they behave (animation).

2.2 Adding geometry to Petri nets

The problem this project is tackled with is simple: We need a way to link the Petri net model to a 3D visualisation, this is, to add extra information so that it can be visualized. Once a Petri net model is created and its real life design is well-designed in the user's mind, what we call a "Geometry" and "Appearance" are created.

For this purpose, there is a need of a geometry editor which will be used to assign a two dimensional location to the elements defined in the Petri net, and of an appearance editor to take care of the shape of the elements.

2.3 Adding appearance to Petri net objects

Once the geometry problem is solved, a shape for each object should be defined. For instance, if places represent tracks, the shape, texture and other attributes should be linked to that object.

For this purpose, there is a need of an appearance editor which will be used to assign 3D visualization features to the elements defined in the Petri net.

2.4 Configuration

Before running the simulation, a definition of how the previous models are connected is needed. This is done in the configuration step as well as the validity check for the Petri net's connections to the geometry.

2.5 Simulating Petri nets

The next step in visualizing Petri nets is add descriptive visuals. With the information provided by the Petri net, geometry and appearance as defined in the configuration file, the simulation is set up.

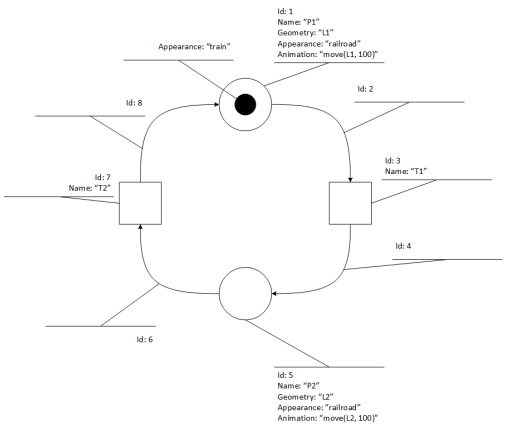

Figure 2: A simple extended Petri net

Using 3D models, textures and animations, the Petri net becomes easier to understand for the user. Continuing with the railway example, tokens become trains that move on tracks, which are places. As the tokens move from place to place, the train is animated in the 3D visualization along the tracks.

Figure **??** shows how a simple 3D simulation of a train track and a train is made out of Petri net model and a simple geometry. Place P1 references L1 as its geometry and also has the shape of a track. The token on place P1 has the shape of a train and it moves with an animation defined by the place P1.

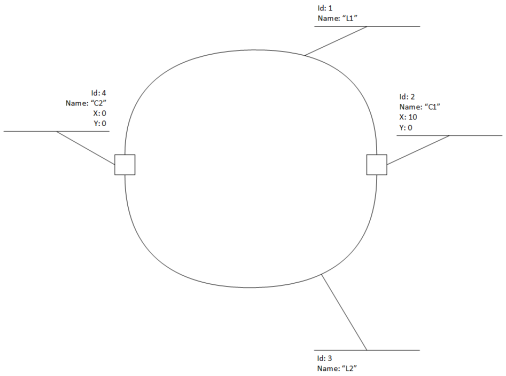

Figure 3: A simple geometry

2.6 General product description

The overall description is a brief introduction to how our software is designed. For this purpose, it contains a description of the actors involved in its use, and also a description of how the software works.

The following persons are all actors of our software:

- Petri net engineer: An engineer whose role is to design and develop a Petri net which suits the company needs.
  For instance, this could be an engineer working at a railway company and in charge of modelling the railway system using Petri nets.
- End user: An actor to whom the Petri net 3D-Visualization would be presented, or an actor who is in charge of presenting the Petri net to another.
  For instance, this could be a manager wanting to see what a Petri net represents.

The software is built around three different concepts:

- Editors: A component responsible for handling the creation and design of one of our models.
  For example: A Petri net editor to create a Petri net model.
  An editor is presented to the user as a GUI in an Eclipse window.
- Simulator: A simulator is, as its name says, a component capable of simulating a Petri net. The simulator is handling all the decisions regarding the Petri net and its behaviour. It then communicates the decisions to a 3D Engine responsible for the visualization.
- 3D Engine: A 3D Engine is responsible for the visualization of a Petri net in three dimensions. It receives input from the Petri net simulator, and also communicates to the simulator when a user performs a certain type of action.
  For example: the user clicks on a specific Place of the Petri net, this triggers a message from the 3D Engine to the Petri net simulator

2.7 Basic functionality

This software allows the engineers to create a Petri net 3D Visualization using a combination of the different editors built for this project. The files created with each of the following editors are to be combined using the configuration editor:
Petri net editor, Geometry editor, Appearance editor.

Once the files are linked together in the configuration editor, a 3D Visualization can be launched in the workbench.

The simulator then initializes the 3D Engine with all the informations needed for the visualization. The 3D Engine then relies on the simulator to know which next move it should perform on the Petri net.
To interact with the simulation, the user can either click or press certain keyboard buttons in the GUI for the visualization.

Figure 4. shows the different components of our system and their connections.

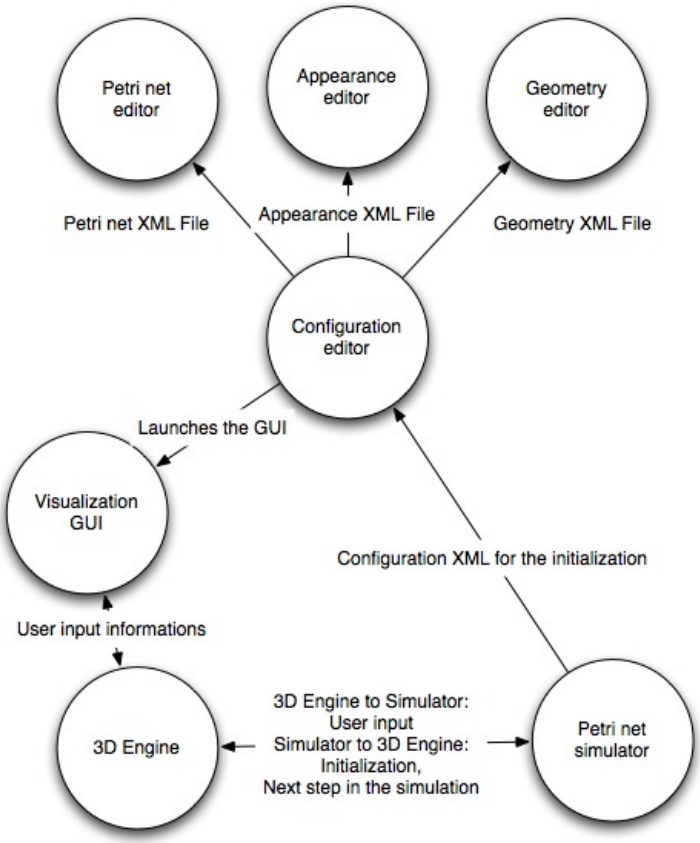

Figure 4: System design

3 System Features

3.1 Petri net editor

Author: ***Albert***

The Petri net editor is a component that will extend the features provided by the *ePNK*, in order to fulfil the requirements of the project. The extended features include *Input Places* and the definition of links between geometry and animation components.

3.1.1 Functional Requirements

1. The Petri net editor **shall** allow the user to create, edit and delete Places.
2. The Petri net editor **shall** allow the user to create, edit and delete Tokens inside Places.
3. The Petri net editor **shall** allow the user to create, edit and delete Transitions.
4. The Petri net editor **shall** allow the user to create, edit and delete Arcs. An arc **shall** connect a Place to a Transition or vice versa.
5. The Petri net editor **shall** allow the user to define a Geometry label to a Place.
6. The Petri net editor **shall** allow the user to define an Appearance label to a Place.
7. The Petri net editor **shall** allow the user to define an Input Place label to a Place.
8. The Petri net editor **shall** allow the user to define an Animation label to a Place.
9. The Petri net editor **shall** allow the user to save and load a Petri net model.
10. The Petri net editor **shall** create a Petri net file in a format that can be read by the Simulator.
11. It **would be nice** that the Petri net editor allowed the user to undo and redo actions.
12. It **would be nice** that the Petri net editor allowed the user to copy and paste.

3.1.2 Use cases

The features are shown in Figure 5.

3.2 Geometry editor

3.3 Appearance Editor

3.4 Configuration Editor

Author: ***Albert***

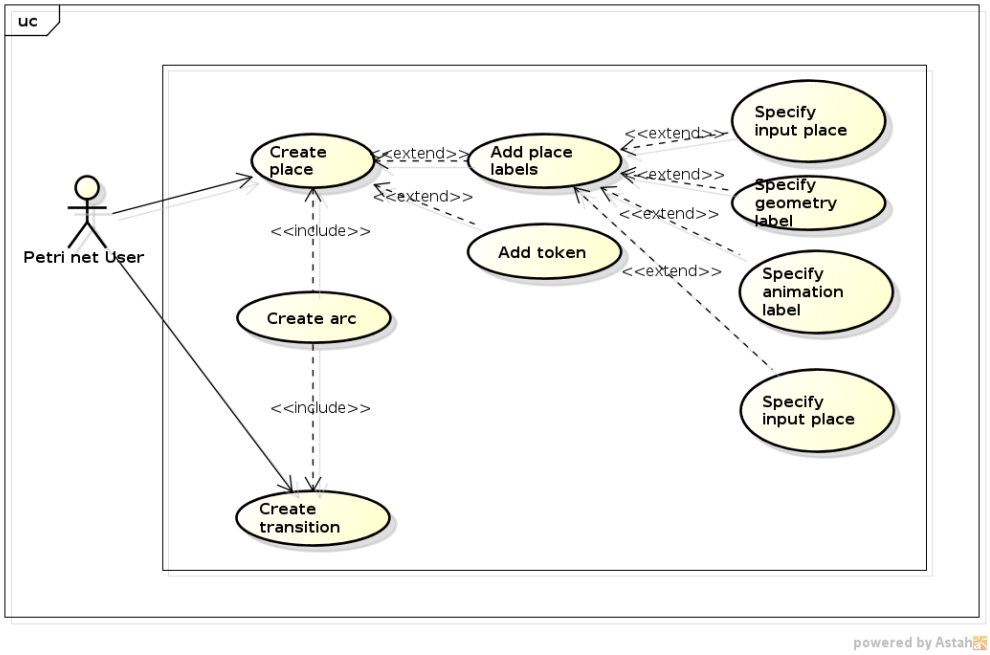

Figure 5: Use cases for the Petri net Editor

The configuration editor is a component that will work as a connector between the Petri net editor (Section 3.1), the Geometry editor (Section 3.2) and the Appearance editor (Section 3.3).

3.4.1 Functional Requirements

1. The configuration editor **shall** allow the user to input a Petri net file containing its model.
2. The configuration editor **shall** allow the user to input a Geometry file containing its model.
3. The configuration editor **shall** allow the user to input an Appearance file containing its model.
4. The configuration editor **shall** allow the user to start a simulation with the referenced Petri net, Geometry and Appearance models.
5. The configuration editor **shall** allow the user to validate the data.
6. It **would be nice** if the configuration editor to save and load a specific configuration to a file.

3.5 Use cases

The features are shown in Figure 6.

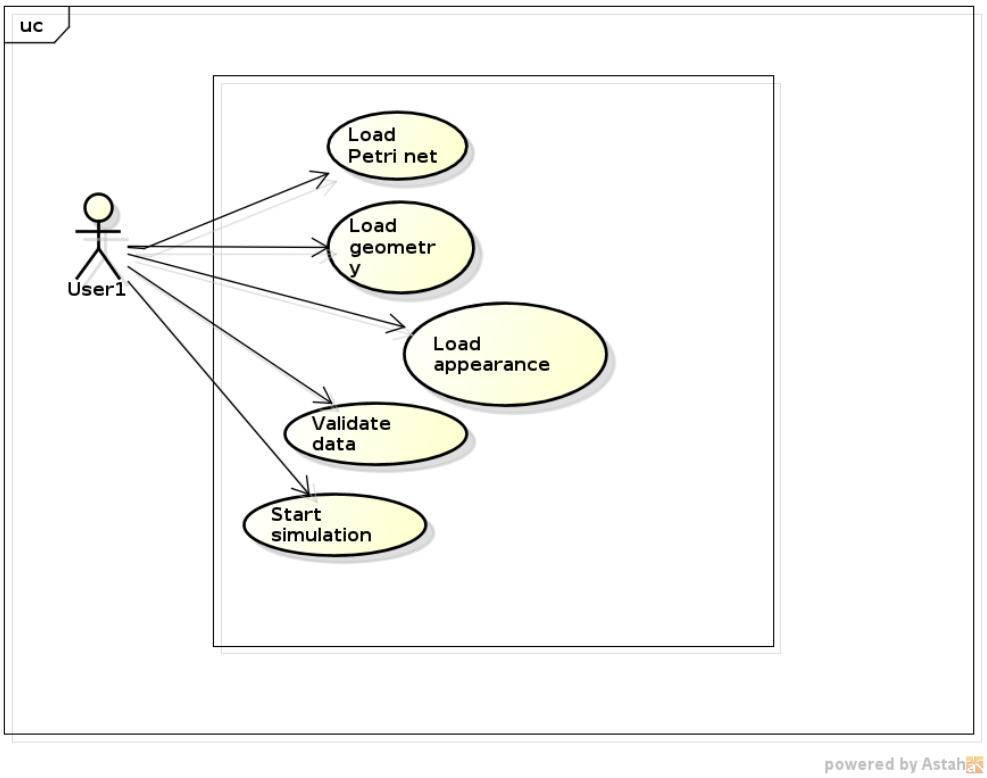

Figure 6: Use cases for the Configuration Editor

3.6 3D Simulator

Welcome to my thing.

4 Non functional requirements

4.1 Implementation constraints

4.2 Documentation

4.3 Quality Assurance

5 User Interface

5.1 Technology

5.2 GUI parts

5.3 Handbook

6 Architecture

The architecture of this **Extended Petri net Simulator** has been designed as modular, meaning that the system is divided in different components which are connected through the defined interfaces.

7 Glossary

GUI Graphical User Interface

Petri net A mathematical and graphical model for the description of distributed systems. It is a directed bipartite graph, in which nodes represent transitions and places. The directed arcs describe which places are pre- and/or postconditions for which transitions. [source wiki]

Synchronization Transition finings are synchronized on the occurrences of external events, such as when animation is finished or user triggers the transition.

Token Petri Net element which moves along the Petri net places through transitions.

Use case A list of steps defining interactions between a role (e.g. "Technical user"), also known as an actor, and a system for achieving a goal. The actor can be a human or an external system

3D Three dimensional.

ePNK Eclipse Petri Net Kernel.