Project Definition
Extended Petri net Simulator

Group A

September 26, 2013

Abstract

This paper is a definition to our Software Engineering 2 project. It will provide an outline of our project, while also introducing concepts and terms going to be used in our Systems Specifications.

Contents

1 Introduction

> 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 get a feel of how the model works to understand and validate a system.

However, this 3D visualisation that we would like to present comes with a problem: 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 work flow 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 can be 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: signified by circles, represent states.
- Transitions: signified by squares, represent conditions.

[1] Petri Nets and Industrial Applications: A Tutorial. Richard Zurawski, MengChu Zhou.PetriNetsandIndustrialApplications:ATutorial.RichardZurawski,MengChuZhou.(http://www.cs.uga.edu/~eileen/WebEffectiveness/Papers/PetrinetsAndIndustrialApplications.pdf)

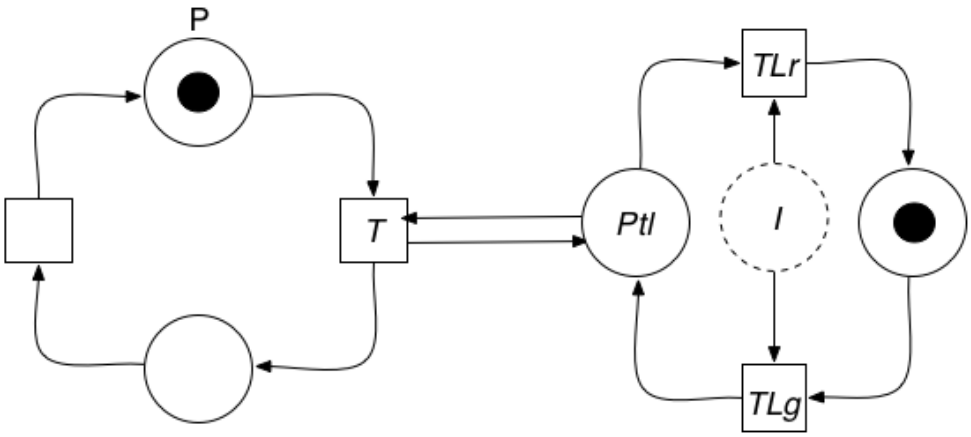

Figure 1: An example of a basic Petri net

- Arcs: indicate which places are preconditions/postconditions for which transitions.
- Tokens: signified 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 1 as a model for a railway with a traffic light. The token in Place P represents a train moving on a railway. When this train arrives to Place P, 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 a user clicks on the Place I, 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. For this purpose, 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" is 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.

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.

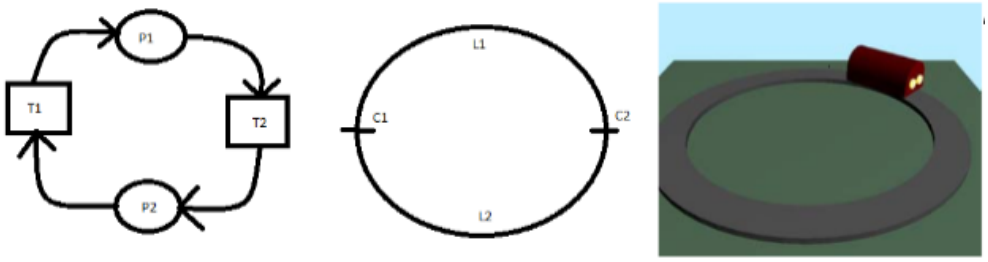

Figure 2: Left: a simple Petri net. Center: a geometry model. Right: A 3D visualization.

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 2 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.

3 Functionality

In order to implement the functionalities of the concepts described in the previous section, editors are needed. Each of these editors will be described in this section.

3.1 Petri net Editor

3.1.1 Purpose

The basis for the main functionality of the software is the Petri net Editor. This editor enables a user to create or edit a Petri net that can be interpreted and used by the other editors.

3.1.2 Users

Users with knowledge of Petri net logic and general technical skills are the main users of the Petri net editor. However, both simple and complex Petri nets can be implemented, so the technical skills of the users may vary accordingly.

3.1.3 Use cases

The main use cases of the Petri net Editor are conveyed in the following diagram:

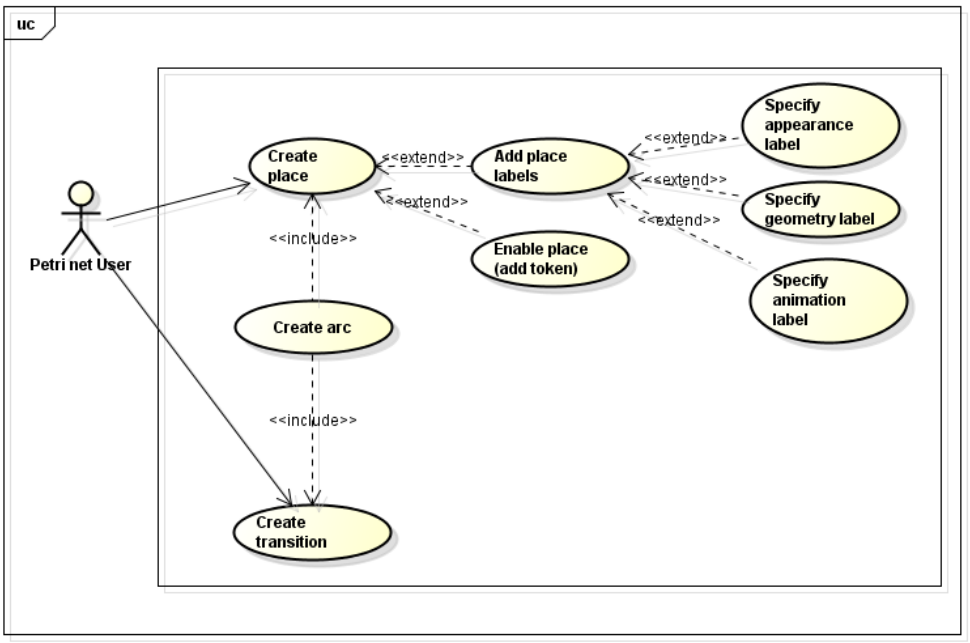

Figure 3: Use cases for the Petri net Editor

The user will be able to create places, arcs and transitions. The user must start by creating a place or transition, arcs can only be created once there is a target and a source. A place or a transition can be dragged from a menu to the canvas, while the arc must be selected in the menu, then dragged from the source to the target. The user will be able to make the arc bi-directional by dragging a new arc in the opposite direction of the existing.

A place can have up to three labels associated: appearance, geometry and animation. These labels are used to control the appearance, geometry and animation in the simulator.

An example of an appearance label is "train", which will tell the simulator to use the appearance linked to "train" for the token(s) and the arcs associated with the place. See "Appearance editor" Section 2.3. An example of an animation label is "fast", which will enable a fast token movement on the place and associated arcs. An example of a geometry label is "line_1", which will link the place to a line specified in the geometry editor.

Finally, a user can enable a place to receive input during runtime; this will enable the user to add a token on that place.

The Petri net editor is linked to the other editors, especially through labels. In the following, the Geometry editor will be described.

3.2 Geometry editor

3.2.1 Purpose

As it has been previously discussed in Section 2, in order to create a 3D representation of the Petri net model, we need to add some geometrical information to it. For this purpose, the user will interact with a geometry editor to define the location of the objects defined by the Petri net editor. These objects on their location will later be displayed in the 3D visualisation model. The geometry editor will allow the user to:

- draw lines - corresponding to places in the Petri net model
- add bend points to lines - for creating curved lines
- draw connectors - corresponding to transitions in the Petri net model
- add input points - corresponding to special constraints for transitions (e.g switches, traffic lights, etc.)

3.2.2 Users

The geometry editor will be used by technical users who are familiar with basic geometrical concepts and have a clear vision on how a geometry model is designed. No other specific previous knowledge is required but comprehension of Petri net models is an advantage.

3.2.3 Use cases

The main use cases of the Geometry editor are displayed in the following diagram:

1. **Create line**: The user drags a line from the editor menu and draws it in the canvas. The number of lines is not restricted so the action can be repeated as many times as it is necessary but it is compulsory to have two connectors created before attempting to draw a line. Lines should also have a label that links back to the Petri net element it refers to.

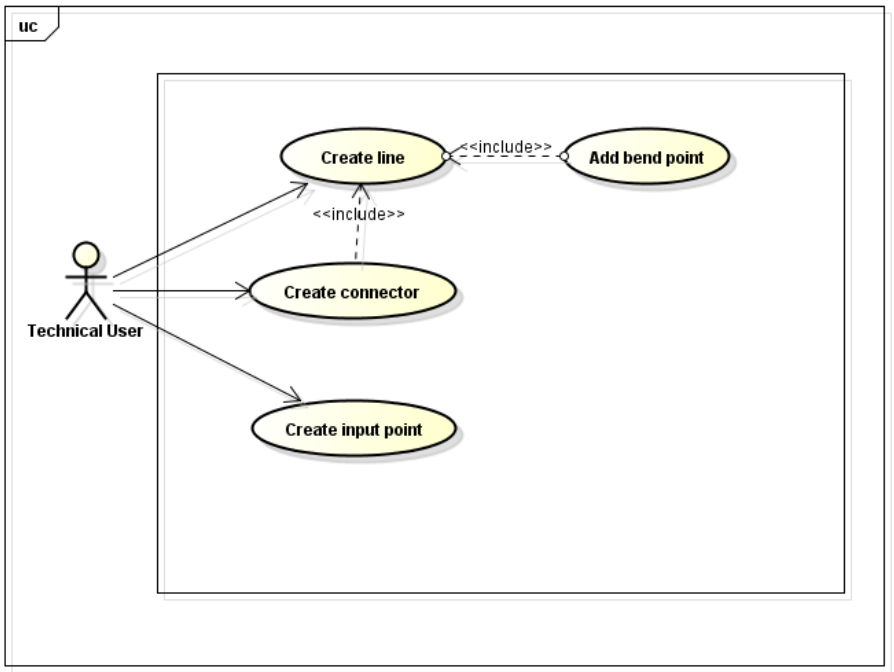

Figure 4: Use cases for the Geometry Editor

2. **Add bend point**: Bend points can only be added after a line has been created by clicking the line in order to create a curve. The number of bend points per line is restricted to one.
3. **Create connector**: Connectors can also be created from the editor menu and drawn in the editor window. Their functionality is to link two or more lines together.
4. **Create input point**: Input points will be created in a similar way from the editor menu and can be placed anywhere in the canvas. However, they will have a label and an appearance attached to them in order to define the link back to the Petri net.

3.3 Appearance Editor

3.3.1 Purpose

The next step in reaching the 3D visualization of the Petri net model is to determine the visual characteristics for each element in the model. For this purpose, the appearance editor is used to define the shape, color, texture and any other information needed for a clear display of the initial model.

3.3.2 Users

The appearance editor can be used by any user both technical and nontechnical as it only implies linking the appearance labels defined in the Petri net model to a file containing all the information related to visual aspects: shape, texture, color. These can be .jpg files or special 3D documents whose structure is yet to be discussed.

3.3.3 Use cases

The main use cases of the appearance editor are conveyed in the following diagram:

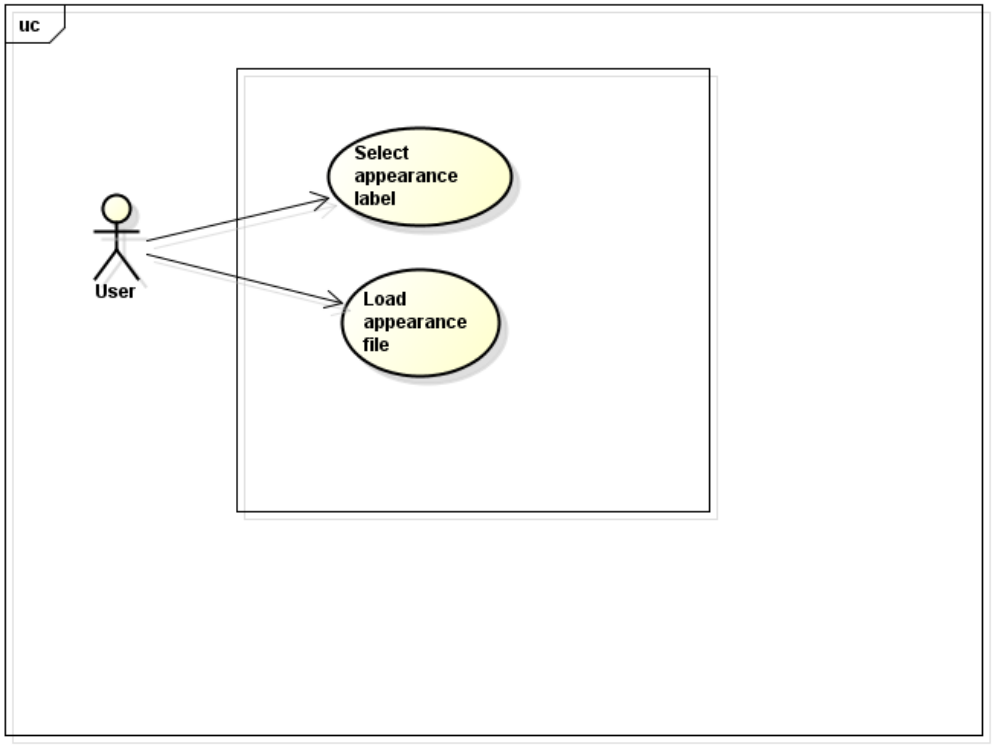

Figure 5: Use cases for the Appearance Editor

1. **Select appearance label**: The user will be able to select one of the appearance labels previously defined in the Petri net editor from a pop-up menu. For example, if the user wants to add appearance information to a token which in the Petri net model has as appearance label "train", then his/her selection from the pop-up menu should also be "train". The next step is described in use case 2.
2. **Load appearance file**: The appearance editor will allow the user to browse among a list of files containing information related to the 3D visualization and load the one corresponding to the previously selected label.

3.4 Configuration Editor

3.4.1 Purpose

The configuration editor is intended to define a link between the Petri net model, the geometry and the appearance information files previously created by the user. Each element in the Petri net should have a corresponding geometry figure and appearance features assigned in order to have a valid configuration file for the simulation.

3.4.2 Users

This editor will be used by all users, both technical and nontechnical as it only implies the association between three existing files. The interface should be intuitive and organised in a way that is familiar to the user in terms of loading files therefore no particular knowledge is required.

3.4.3 Use cases

The main use cases for the configuration editor are conveyed in the following diagram:

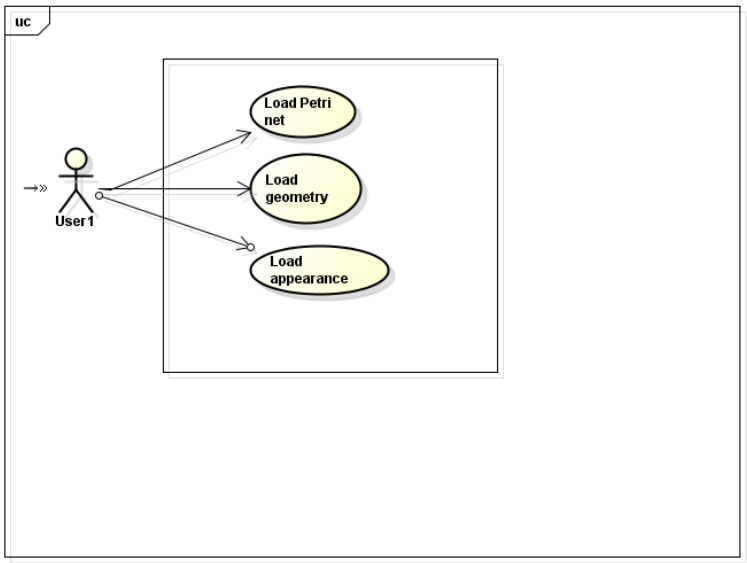

Figure 6: Use cases for the Configuration Editor

1. **Load Petri net**: The user will be able to browse and load a Petri net model file designed using the Petri net Editor.
2. **Load geometry**: The user will be able to browse and load a geometry file designed using the Geometry Editor.
3. **Load appearance**: The user will be able to browse and load an appearance file designed using the Appearance Editor.

3.5 Simulator

3.5.1 Purpose

Once the Petri net Editor and Geometry Editor have been set up correctly the user will be able to run the 3D simulation based on the information from the Appearance and Geometry Editor. During the 3D simulation the user will be able to add or remove trains, control traffic lights and control railroad switches through a Graphical User Interface (GUI).

3.5.2 Users

The simulator can be used by any user, both technical and nontechnical as they both have an interest in seeing the 3D visualization of the work done in the editors. Users do not need to have any prior knowledge about Petri nets since the simulation is controlled with a GUI and has no visual connection to the Petri net.

3.5.3 Use cases

The main use cases for the Simulator are conveyed in the following diagram:

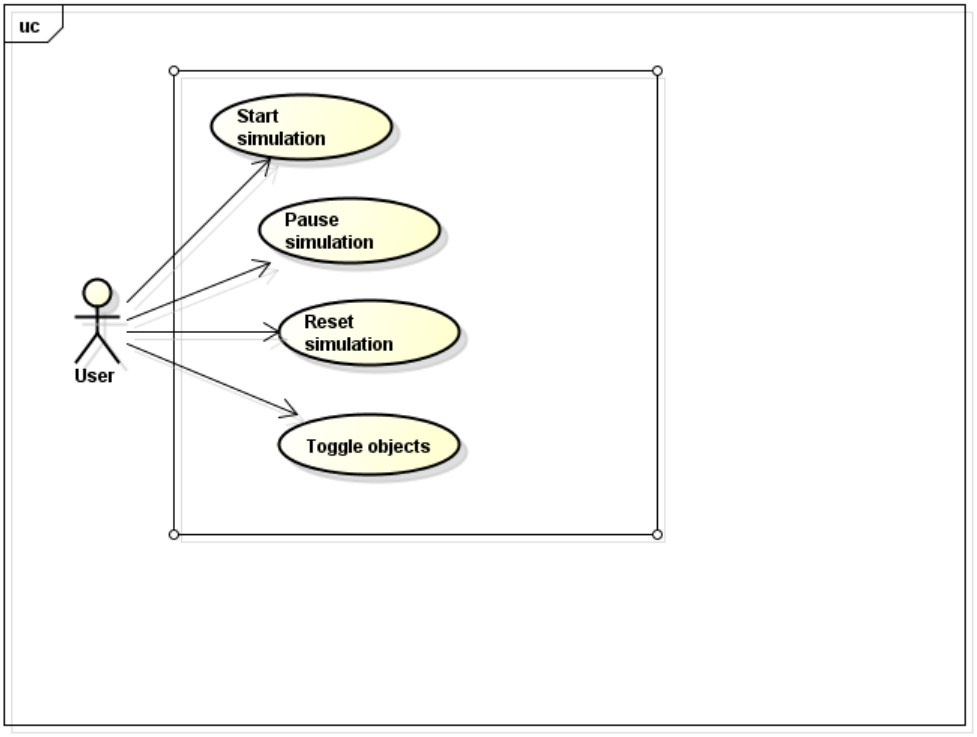

Figure 7: Use cases for the Simulator

1. **Start, pause and reset**: The user is able to start, pause and resume the simulation with 2 different buttons: one for play and pause, and another for resetting the simulation.
2. **Toggle objects**: The user can toggle an object, for example a traffic light, in the simulation by clicking on it; switching (e.g.) a light from green to red or the other way around.

4 Platform

4.1 Requirements

The Petri net simulator runs with extensions in the Eclipse Kepler IDE , and is written in Java. The Eclipse platform is open source and free. Eclipse is cross-platform, and runs on Linux, Mac OS X, Solaris and Windows.

Any modern PC should be able to run the Petri net simulator easily. The recommended minimum resolution is 1366x768 or equivalent. A mouse and keyboard is also recommended.

5 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.