Handbook
ePNS

Group E

Autumn 2012

Contents

1 Introduction

Author: ***Anders***

This is a handbook for the “group **e** **P**etri **N**et **S**imulator”: **ePNS**. The purpose of **ePNS** is to facilitate track based simulations of Petri nets. A track based simulation is understood as a simulation of a world, where objects move along tracks, following a set of rules given by the Petri net. The objects can be simulated as trains on tracks, cars on roads, or any other object following a predefined route.

1.1 What can be simulated?

Petri nets are, with the addition of inhibitor arcs[1], Turing complete, meaning that they can implement any computer program. We will not formally prove this property, as it is out of scope for this assignment. But it is not hard to understand why: the inhibitor arc makes it possible to implement a NAND gate (see figure 1). All binary operations can be expressed using the NAND operation, some examples:

$$\begin{aligned}
\overline{A} &= \overline{A \cdot A} && NAND(A,A) \\
A \cdot B &= \overline{\overline{A \cdot B}} && NAND(NAND(A,B), NAND(A,B)) \\
A + B &= \overline{\overline{A + B}} = \overline{\overline{A} \cdot \overline{B}} && NAND(NAND(A,A), NAND(B,B)) \\
A \otimes B &= A \cdot \overline{B} + B \cdot \overline{A} = \overline{\overline{A \cdot \overline{B}} \cdot \overline{A \cdot B}} && \text{use previous formulas} \ldots
\end{aligned}$$

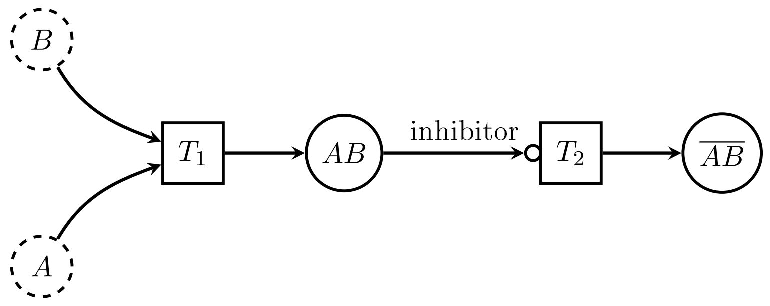


Figure 1: Petri net for a NAND gate

Consequently not all Petri nets are suited for track based simulations. The main concept of Petri nets are that transitions consume tokens from places connected with inbound arcs, and produces tokens in places connected with outbound arcs.

Therefore don’t expect any Petri net to produce a logical or useful 3D simulation. Not even if it doesn’t include inhibitor arcs.

In **ePNS**, if a token is produced in a place associated with a track, the token itself will be associated with a moving object. Therefore, Petri nets that produce the same amount of tokens as it consume

[1]Inhibitor arcs prevents transitions from fireing unless the associated place is empty. Inhibitor arcs are not implemented by **ePNS** yet.

will be most suited for **ePNS** simulations. This is not a mandatory property, but it ensures that the moving objects don't disappear or appear out of the blue[2].

Other places can be associated with objects that show the state of the underlying logic. In the train simulation world, they could be traffic lights or switches.

However, other places should be designated to receive input from the user. Such places might not be part of the initial logic, but should be added where input is needed.

1.2 How to use the handbook?

In the Installation section, the necessary steps to have a running installation are reviewed. An installation includes the Eclipse IDE and additional modules.

For first time users, the section 3 (Tutorial) will cover the steps needed to create a simple Petri net simulation. These steps include:

- Creating the Petri net.
- Laying out the tracks and the positions of input and output objects in the associated geometry model.
- Selecting the appearance of the elements.
- Creating the configuration for the simulation.
- Running the simulator.

For more experienced users, the section 4 (User guide) will provide detailed information on the various steps in creating more complex simulations.

Please consult the index and glossary for information on specific subjects.

1.3 Basic functionality of ePNS

Author: ***Cosmin***

In order to use **ePNS**, the application user starts by using the **Petri net Editor** to create and configure the Petri net and the **Animations** that are executed during the simulation. These will be later loaded by the **Graphical Simulator**, which, after being started, will simulate the movement of tokens through the Petri net and will execute the configured animations, resulting a visual representation of the Petri net's simulation on the screen.

Then, using the **GeometryEditor**, the **Geometry** would be created. It would be later used by the **Graphical Simulator** in order to know how (on which paths) to move objects/token representations and where to place different objects in the 3D space[3].

[2]In a production line simulation it might be on purpose. For example, four wheels and a car body disappears and a car arises.

[3]Even though the geometry is specified in a 2D space, during the simulation, all the object representations will be drawn as 3D objects moving on a plane.

The **Graphical Simulator** also requires information about the **Appearance**, in order to know how to represent tokens, tracks and other objects during the simulation. It is a simple editor that connects labels (keys) with 3D Models(vrml, png, jpg ...), textures or just plain data (Colors, Shapes).

The last step is to create a **Configurator** that connects the previously created configurations and allows the user to start the graphical simulation. When started, the **Graphical Simulator** reads the state of the simulation from the *Petri net*. This read state does not include exact positions of tokens in space, this information being loaded from the *Geometry*, or appearance information, loaded from the *Appearance*. After initialization, the Graphical Simulator displays the state of the simulation and handles all the users' interaction as specified in the rest of this document.

For further details of what exactly each of the components allows the users to do please check the following section or, in order to get more details, regarding implementation of **ePNS**, please read the *System Specification*.

1.4 General concepts

Author: ***Cosmin***

This subsection will introduce the general concepts used in the **ePNS**system. More details are provided in the *System Specification*, however the most important concepts are presented below.

First of all, the classical concepts of **Petri nets** have been extended to accommodate the required information for the graphical visualization of the simulation:

Input Places - in order to provide the users with more power and customizability, some of the Places in the Petri net can be configured to allow users, during the graphical simulation, to drop (create) tokens. These are called *Input Places* and act as normal Places in all other respects, except for that they permit the possibility of a token being created there. For example, in a train track simmulation, it allows the creation of simulation features such as a Traffic lights or switches with which the users can interact during the graphical simulation.

Animations - can be associated by the user to a particular Place and are run when a Token is added on that Place, either by result of executing a transition or by being dropped there (after a user interaction). Even though the token is removed from the source Places of the fired Transition, they are not available for firing a new Transition until the animations associated with a place are finished. More details will be specified later, but the supported animation types include: moving an object on a path, showing or hiding objects, wait a fixed amount of time.

Place Appearance - each Place can have an associated appearance, describing how it must look like in the Graphical Simulator.

Token Appearance - each Token can have an associated appearance, describing how it must look like in the Graphical Simulator. Thus, the appearance of a Token will not change based on a place. This will allow multiple tokens, with different representations, to be on the same place/track.

Arc Identities - each arc can have attached an identity used to control the flow of tokens (or, more precisely, of token representations) in the simulation. For e.g., if we have a Transition with one input Arc and two output Arcs and we take the case of simulating a train running on a track, using the same identity on the input Arc and on one of the ouput Arcs will tell the Graphical Simulator to move the Token representation (a train), which came on the input Arc, on the corresponding output Arc. This allows a token representation to move continously in the direction the user wants, without being destroyed or unnecessarily recreated.

Regarding the **Geometry**, as defined, it allow the users to specify the positions of objects and the paths on which they move in the simulation space. The most important related concepts that need to be presented at this point are:

Track - defines how a curve (or line), on which an animation can take place, looks like. It can also be connected with information about what the surface of this track looks like and usually are used as graphical representations of Places.

Simple Position - defines just a position in the simulation space and can be connected to the an appearance it has. Can be used for completing the specification of some animations, for representing an Input Place or just for displaying simple objects during the simulation.

Referring to the **Appearance**, it allow the users to easily define how objects look like during the simulation. In **ePNS**, there are mainly two big types of appearances that can be configured:

Shape - defines how a 3D Object displayed in the simulation should look like. For example, it can be a reference to a file storing a 3D Model, which can then be loaded in the application or it can simply be a 3D Object, such as a Cube or Sphere.

Surface - defines how a surface displayed in the simulation should look like. It can be applied, for example, to a train track, and it could be either just a Color or a reference to a file containing a texture that can be applied on the surface.

2 Installation

Author: ***Cosmin***

This section discusses the installation of the product and presents the requirements and main steps necessary to make the system work.

2.1 Prerequisites

Our product is built as a series of extensions for the Eclipse Development Platform and requires Java. Furthermore, ePNK 0.9.4 has to be set up in Eclipse and the Java3D 1.5.1 library has to be installed.

First of all, *Eclipse 4.2.0* (Juno) has to be installed on the computer. The installation can be done by following the instructions found in the Eclipse Install Guide [4] or by navigating to `http://www.eclipse.org/downloads`.

Secondly, *Java Runtime Environment 1.6* has to be installed. Setting up the Java environment can be done by following the instructions found on the Java homepage, `http://www.java.com` or by navigating directly to the Java Downloads page [5].

The *ePNK* can be installed, as an Eclipse extension, by following the details provided on its homepage, `http://www2.imm.dtu.dk/~eki/projects/ePNK/install-details.html`. For more details, ePNK's manual [6] can also be consulted.

The last requirement, *Java3D* library, can be set up by following the instructions on the product's home page, `http://java3d.java.net/` or on the install guide page [7].

2.2 Product installation

As mentioned before, the product has to be installed as an Eclipse plugin. The final version of the product will be available as an Eclipse plugin that can be installed via an update site (or, for offline installation, as a site of files that need to be copied in the Eclipse plugins directory). The plugin will be available only after Eclipse has been restarted, if it was running.

During the development period, the product is available as a set of projects[8] that have to be downloaded locally and imported into the Eclipse Development Environment. After opening the projects, a Runtime Eclipse Workbench has to be started by selecting one of the projects and then by clicking on the *Run* command.

[4] http://wiki.eclipse.org/Eclipse/Installation
[5] http://java.com/en/download/manual.jsp
[6] http://orbit.dtu.dk/getResource?recordId=275136
[7] http://download.java.net/media/java3d/builds/release/1.5.1/README-download.html
[8] The latest version is available at https://svn.imm.dtu.dk/se2/svn/e12-groupE/project/

3 Tutorial

Author: ***María***

This section is a practical guide that will show you the basic functionality of the ePNS Petri net simulation system; the tutorial will go through a simple example to allow you to learn how to use it.

Imagine that you want to simulate a scenario in which a red cube follows a circular line when you press a button. First, you need to model the corresponding Petri net where places are visually represented by an image in a 2D plane (in this case, the line and the button), and tokens can represent 3D objects moving on top of the 2D image (in this case, the cube following the line). Transitions have no visual representation, but they are of course still necessary for the proper performance of the Petri net. The model designed for this scenario could be the one shown in Figure 2.

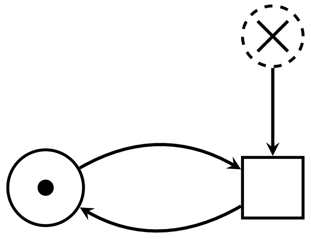


Figure 2: Petri net model

The place on the left is the one that will represent the circular line, and the one with the dashed line will represent the button. Note that this second place is not an ordinary place: it is an *input place* , where tokens can be externally inserted during the simulation. When the button is clicked, a token will appear in that input place, and the transition will be fired. In an ordinary Petri net, when this happens the tokens in the transition's source places are removed from said places and new ones appear in the transition's target places. However, **ePNS** offers the posibility of putting identities to arcs to prevent tokens from being destroyed in the process and preserve its appearance (if it's a cube, a train, a car...) during the simulation.

The model can be created using the Petri net editor with **ePNS**'s custom Petri net type (ePNS Petri net) . For this purpose, you have to create a PNML document and edit it, creating the places, transitions and tokens, set their attributes and conect them accordingly with the help of arcs. To do this, you must follow these steps:

1. In the runtime workbench, click "File > New > Project > General > Project". Click on "Next", give it a name and click "Finish".
2. Right click on the created project, select "New > Other > ePNK > PNML Document" and click "Next" .
3. The project folder should already be the parent folder. If not, change it to make it so. Give a name to your new file and click on "Finish" (note that you should keep the *.pnml* extension.

4. Open the file and you will see a tree editor. Expand the root node and right click on "Petri Net Doc". Add a Petri net by selecting "New Child > Petri net (ePNS Petri net)" (See Figure 3).
5. In the newly created Petri net, right click and select "New Child > Page" (See Figure 4).
6. Double click on the created Page to open the graphical editor.
7. In this editor, you can add Nodes (Places and Transitions) by clicking on the corresponding symbol in the *Palette* on the right hand side of the screen, and clicking on the canvas to place it. (See Figure 5).
8. You may connect Nodes with the Arc tool in the Palette. To do this, select the Arc icon, click on the desired source Node and drag on top of the desired target Node.
9. To modify the attributes of an element, right click on it and click on "Show Properties View". The selected element's properties can be changed in this view.
10. Every element has an attribute called "Id". Change it to be able to differentiate each component. For example, assign "P1" to the classical Place, "P2" to the Input Place, "T1" to the Transition, "A1" to the Arc connecting the Input Place with the Transition and "A2" and "A3" to the other two Arcs.
11. Places have an attribute that indicates if they are Input Places. Edit it in each Place to give it the correct value (true or false).
12. The aforementioned Identity attribute of Arcs can be changed in the Properties View. Set the identity of "A2" and "A3" to 1. You will see that the color of the Arc changes according to its Identity (in this case the arcs will turn red).
13. When you have finished the structure, you must add the Token. To do this, select the Label tool from the Palette and place a Label on the canvas. Use then the Link Label tool to attach the Label to the ordinary Place (in our case, Place P1) and select "Token" (See Figure 6).

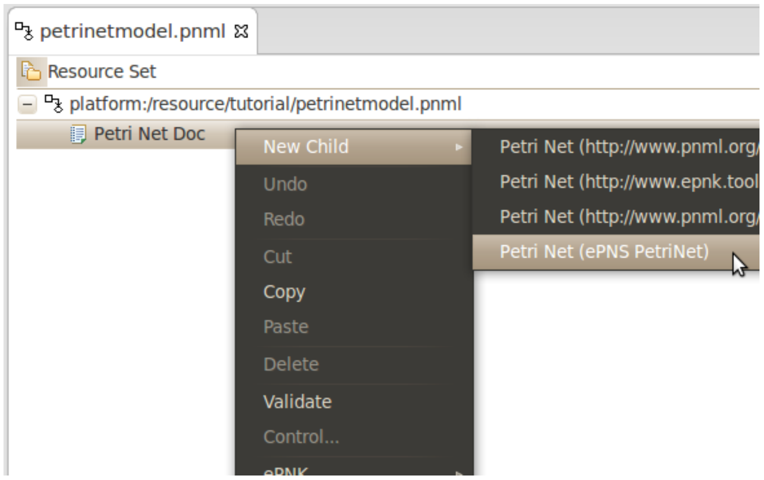

petrinetmodel.pnml
Resource Set
platform:/resource/tutorial/petrinetmodel.pnml
Petri Net Doc
New Child
Undo
Redo
Cut
Copy
Paste
Delete
Validate
Control...
ePNK
Petri Net (http://www.pnml.org/
Petri Net (http://www.epnk.tool
Petri Net (http://www.pnml.org/
Petri Net (ePNS PetriNet)

Figure 3: Creation of the Petri net model

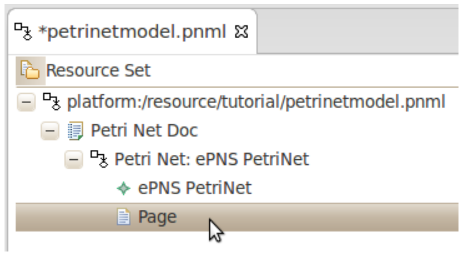


Figure 4: PetriNet Page

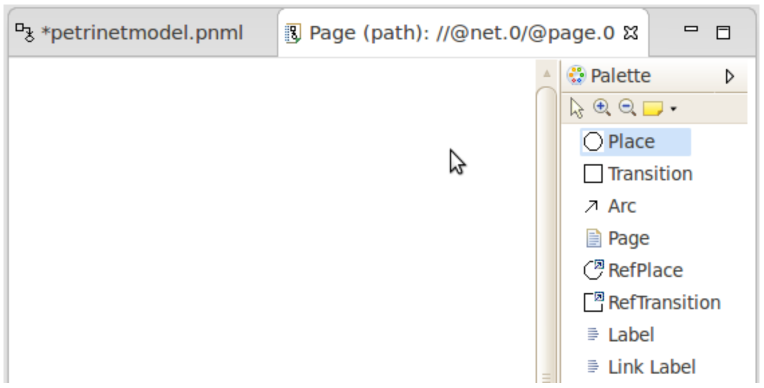


Figure 5: Palette

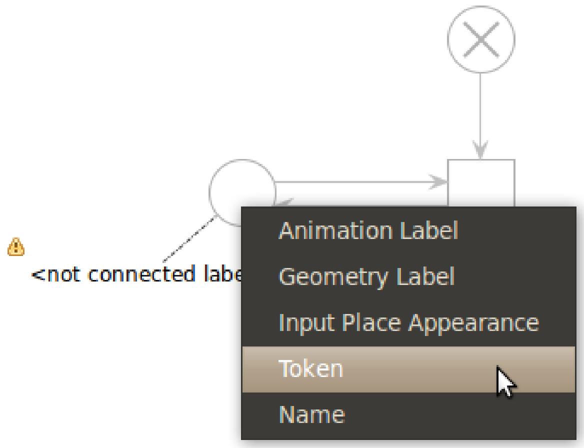


Figure 6: Creation of tokens

You should now have a Petri net model similar to the one in Figure 7.

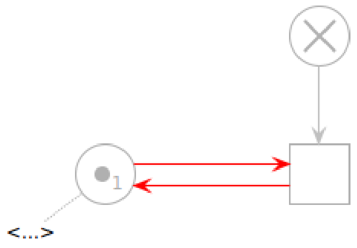


Figure 7: PetriNet model

You can now proceed to the Geometry editor , where you will draw the 2D path along which the Token representations (the cube in this example) will move. The Geometry for this particular example will consist of a circular line and a separate point (where a button that represents the Input Place will appear in the final visualization to allow the user to create a new Token in the Input Place to make the Transition fire).

The following steps describe how to do this:

1. Select the project you created before in the Project Explorer and right click on it to select "New > Other > ePNS > Geometry Diagram". Click "Next".
2. Once again, the project you created should be selected as the parent folder (if it is not, change it). Give your Geometry Diagram a name (keeping the *.geometry_ diagram* extension). Click "Next".
3. The Geometry Model needs also to be created together with the Geometry Diagram. Select your project as the parent folder and give your Geometry Model a name (keeping the *.geometry* extension). Click "Finish".
4. The file with the *.geometry_ diagram* extension is the graphical editor. Open it to start building your Geometry.
5. Create a Track Position on the canvas (in this example, this Track Position will represent the beginning and end of the same Track, the only one used, but it could be the beginning of a Track and the end of a different one; for more information see 4.3.2), select the Track tool and click on the Track Position you just created. A Track with the shape of a square will appear. You can click on the Track and modify its shape to create a circle.
6. Give a name to the Track by filling the Label attached to it, for example "circle".
7. To create the button that will allow the cube to move, you have to add a SimplePosition (for more information on SimplePosition objects, see 4.3.2) to the canvas. Give also a name to this element, for example "point".

The Geometry model should now look like Figure 8, where the empty square is the Track Position, the Track symbol with the word "circle" is the Label and the square with the dashed line and the word "point" is the SimplePosition.

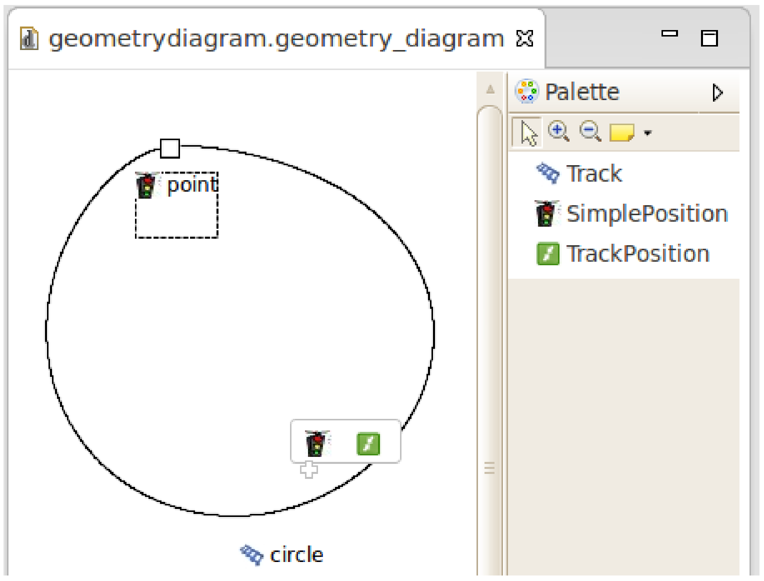


Figure 8: Geometry model

Now that you have the Geometry model, you have to specify which Geometry object corresponds to each Place. For this purpose, you have to assign a Geometry Label to each place of the Petri net model.

1. Go back to the Petri net graphical editor, create a Label and link it to a Place. You will see that you can set the Label to the type "Geometry Label" (See Figure 9), where you must write the name of the correponding Geometry element.
2. Add a *Geometry Label* to the ordinary Place and call it "circle" (See Figure 10).
3. Also add a *Geometry Label* to the Input Place and assign the name "point".

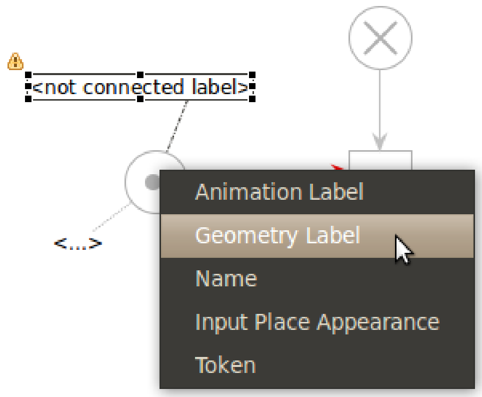
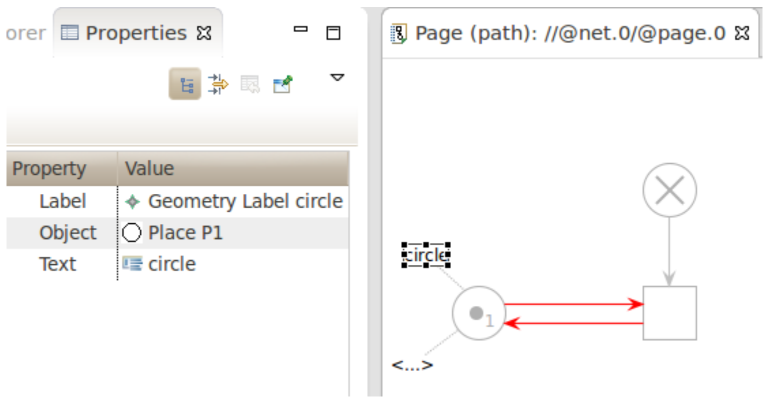


Figure 9: Create Geometry Label



Property	Value
Label	Geometry Label circle
Object	Place P1
Text	circle

Figure 10: Geometry Label of the ordinary place

Now, you need to indicate the behaviour of the simulation: when you click on the button, a red cube shall move following the circular line at a certain speed. This is done by assigning an Animation to the ordinary Place.

1. In the Petri net graphical editor, create a Label and connect it to the ordinary Place, selecting "Animation Label" (See Figure 11).
2. As the Label needs a correct Animation, a warning symbol will appear until you write it. To do that, edit the Label and write "move(3.0)" on it. This is to specify that the cube shall move at a speed with the value 3, and it will move along the Track named "circle" because that is the name of the previously created Geometry Label.

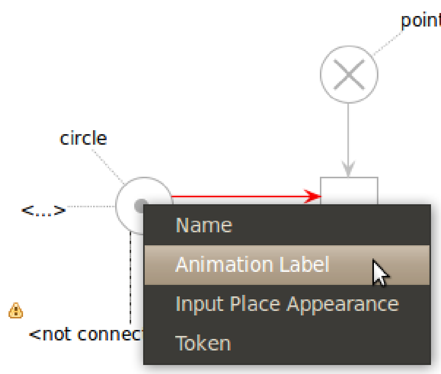
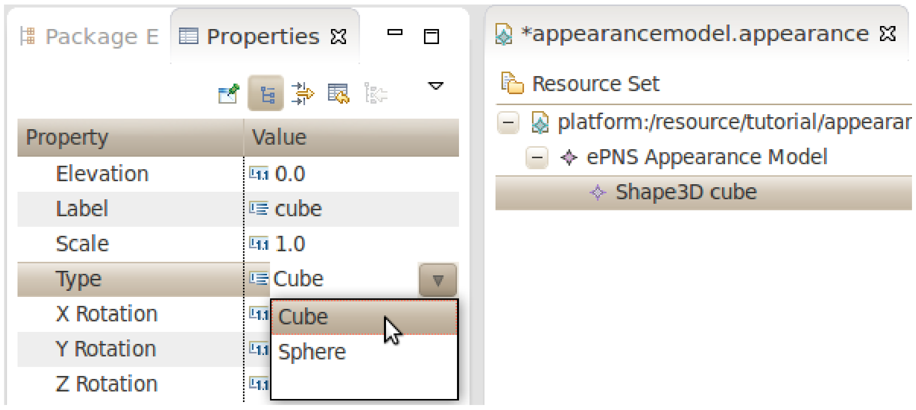


Figure 11: Create Animation Label

The Simulator does not know yet how the different elements of the Petri net model should look like. In particular, the Appearance of the Token (a red cube), the Input Place (a button) and the ordinary Place (a track) should be set. To do this, an Appearance configuration has to be created.

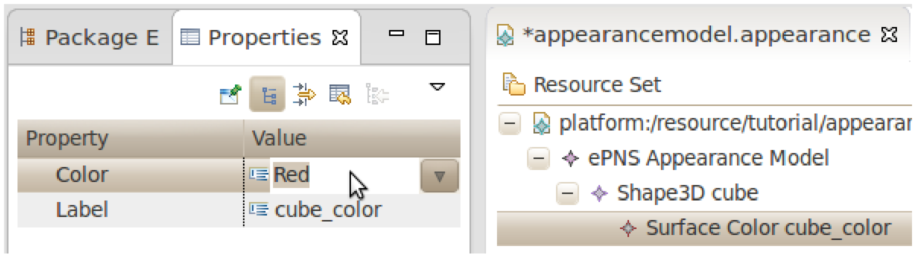
1. Select the project you created before in the Project Explorer and right click on it to select "New > Other > ePNS > Appearance Model". Click "Next".

2. Once again, the project you created should be selected as the parent folder (if it is not, change it). Give your Appearance Model a name (keeping the *.appearance* extension). Click "Next".
3. Select "ePNS Appearance Model" as your Model Object and click on "Finish".
4. Open the file and expand the root node of the tree editor. Right click on "ePNS Appearance Model" and select "New Child > Shape3D". A Shape3D object will appear.
5. Open the Properties View of the newly created shape and fill the Label property with a name, for example "cube". In the Type property you will see that you can choose between some different shapes. Select "Cube" (See Figure 12).
6. To set the color of the shape, right click on its node and select "New Child > Surface Color". In the properties view, modify the Label to put a name and select "Red" for the Color attribute (See Figure 13).
7. To enhance the way the cube shape will look in the visualization, set the Scale attribute to "10.0" and the Elevation attribute to "9.0".
8. You also need a shape to model the button. If you have a predefined model, you can use it to represent the button (otherwise you can simply use a Shape3D as done in the previous step). To load an existing model, you need to:
   (a) Add the model to the Eclipse Workbench by simply dragging and dropping to the project folder (See Figure 14).
   (b) Create a new Model3D (right click on "ePNS Appearance Model" and then select "New Child > Model3D") and assign a name to the Label attribute (for example, "button")
   (c) Right click on the Model3D and select "Load File...". A resource browser will appear, showing your project folder (See Figure 15). Explore it and select the model for the button you added before (the extensions supported are .obj and .3ds). Click OK.
   (d) You may need to change the other attributes of the Model3D (elevation, scale and rotations) later, until it looks nice in the simulation.
9. Finally, you need to set the appearance of the track. You can simply choose a color by creating a new Surface Color, or you can also select a predefined Texture. To do this:
   (a) Drag and drop the texture file (.jpg and .png extensions supported) to the project folder.
   (b) Create a new Texture (right click on "ePNS Appearance Model" and then select "New Child > Texture") and assign a name to the Label attribute (for example, "rail").
   (c) Right click on the Texture and select "Load File..." to choose the corresponding texture.
10. Now that you have finished the Appearance Model, right click on the root node and select "Validate" (See Figure 16). A message saying "Validation Completed Succesfully" should appear.



Property	Value
Elevation	0.0
Label	cube
Scale	1.0
Type	Cube
X Rotation	
Y Rotation	
Z Rotation	

Figure 12: Creation of a cube shape



Property	Value
Color	Red
Label	cube_color

Figure 13: Assigning an color to a shape

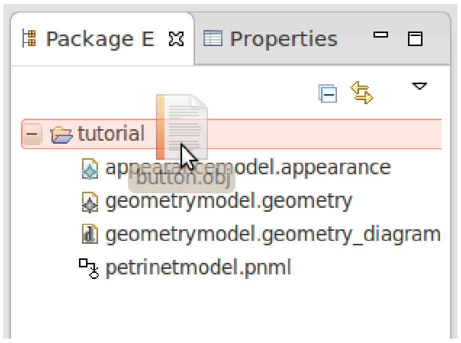


Figure 14: Dragging and dropping a model file to the project folder

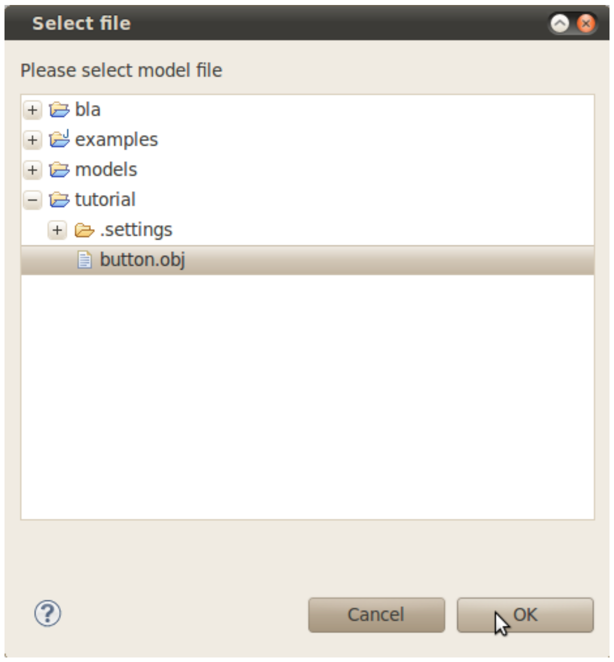


Figure 15: Selecting a model file for the Model3D

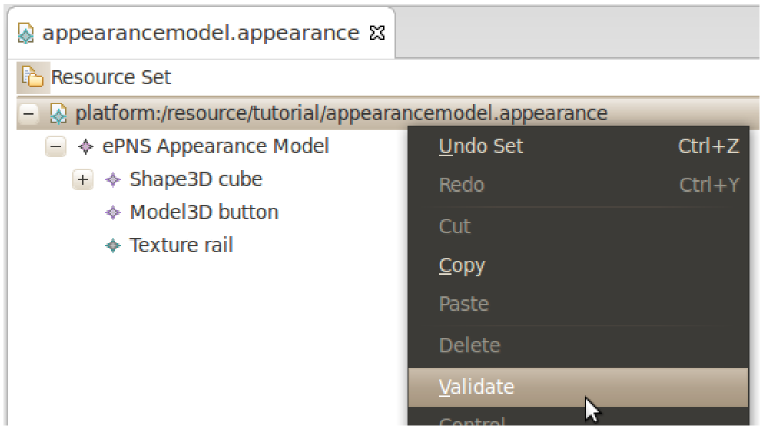


Figure 16: Validating Appearance Model

Now that all the necessary Appearances are created, you need to connect them to the corresponding elements. The appearance of the Tokens are specified in the Petri net Model; the remaining Appearances (what Tracks and SimplePositions look like) should be set in the Geometry Model.

1. To specify that the Token should look like a cube, you need to go back to the Petri net graphical editor. Edit the Label of the Token you created in the ordinary Place, assigning the name of the previously created shape ("cube") (See Figure 17).
2. To set the shape of the button, go to the Geometry graphical editor and modify the Appearance attribute of the Simple Position you created before (the one with the name "point"). Assign the same name as the corresponding Appearance ("button"). (See Figure 18).
3. To set the Appearance of the Track, modify the Appearance attribute of the Track "circle" to assign the same name of the Appearance ("rail").

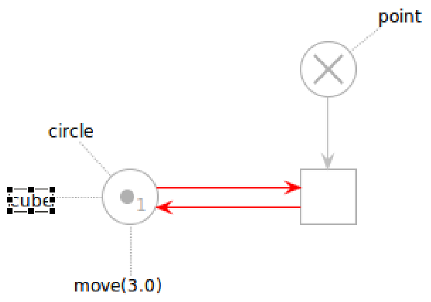
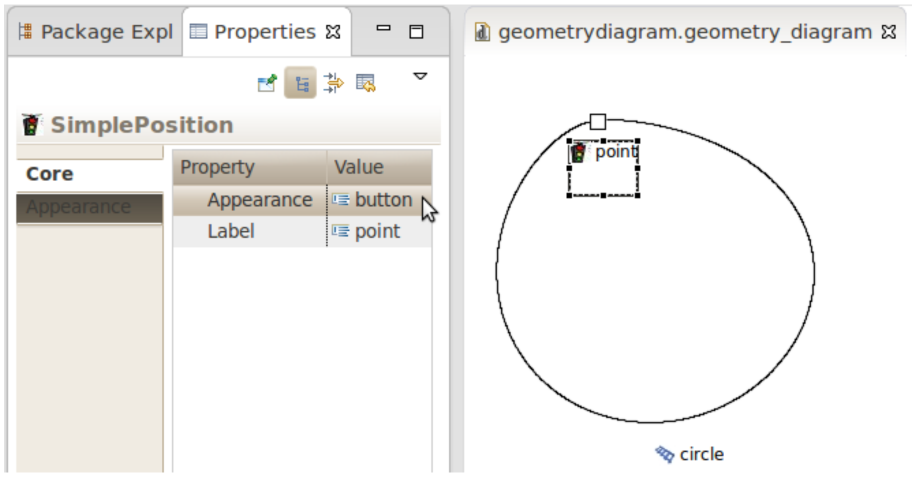


Figure 17: Assigning "cube" appearance to the token



Property	Value
Appearance	button
Label	point

Figure 18: Assigning "button" appearance to the simple position

The models are finished now. You have already checked that the Appearance Model have no errors, but you have to do the same with the Geometry and the Petri net Models (right click on the root node of the tree editor and click "Validate"). You will get an error in the Petri net Model because the IDs of the Petri net and the Page are not set. Modify them in the Properties View, and validate again. This time the validation will success.

Once you have configured the models and the simulation behaviour, you need to connect all the models using the Configurator.

1. Select again the project you created before in the Project Explorer and right click on it to select "New > Other > ePNS > Configurator Model". Click "Next".
2. Once again, the project you created should be selected as the parent folder (if it is not, change it). Give your Configurator Model a name (keeping the *.configurator* extension). Click "Finish".
3. Open the file and right click the root node of the tree editor. Select "Load Resource...".
4. A new window will appear where you can select the files you want to connect (See Figure 19). Click "Browse Workspace...", look for the project you are working in and select the *.pnml* file you created before. Click OK twice to confirm your choice.
5. Repeat the previous step to add the *.geometry* and *.appearance* files.
6. Once you have loaded the three files as resources, you need to attach them to the Configurator. To do this, open the Properties View of the Configurator object and edit its three properties selecting the corresponding model (in our case, "ePNS Appearance Model" for ePNS Appearance, "ePNS Geometry" for ePNS Geometry and "Petri Net Doc" for ePNS Petri net) (See Figure 20).
7. Finally, you can set a default track width for the visualization. Modify the attribute of the Properties View and write "5.0".

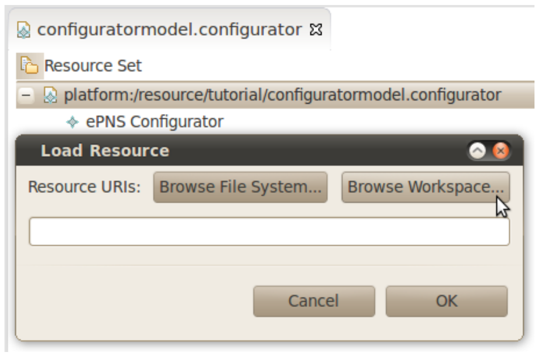
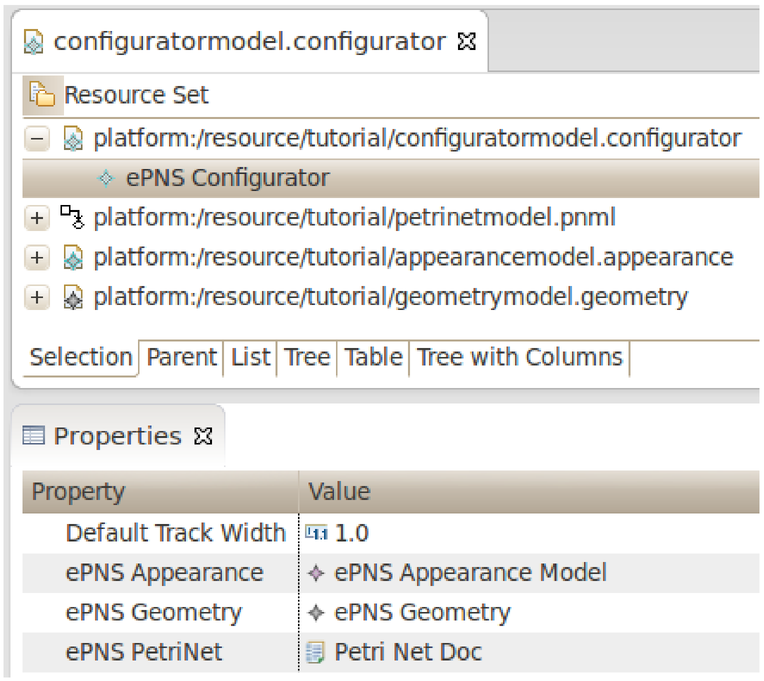


Figure 19: Loading resources


configuratormodel.configurator

Resource Set
- platform:/resource/tutorial/configuratormodel.configurator
  - ePNS Configurator
- platform:/resource/tutorial/petrinetmodel.pnml
- platform:/resource/tutorial/appearancemodel.appearance
- platform:/resource/tutorial/geometrymodel.geometry

Property	Value
Default Track Width	1.0
ePNS Appearance	ePNS Appearance Model
ePNS Geometry	ePNS Geometry
ePNS PetriNet	Petri Net Doc

Figure 20: Configurator properties

By this point, the simulation is ready to be visualized. To start it you must right click on the Configurator object and select "Start simulator".

After starting the Simulator, a new window like the one shown in Figures 21 and 22 will be displayed. Interacting with the simulation can be done in the following steps:

1. To start the Graphical Simulation animations, press the Start Button on the left side of the window.
2. In order to stop or pause the animation at any point, use the Stop/Pause/Resume[9] buttons available on the top of the window.
3. Adding a new token in the place set us Interactive Input can be done by just clicking on its representation.

[9]Please note that Resume is only available after Pause has been clicked, and Stop is only available after Start has been clicked.

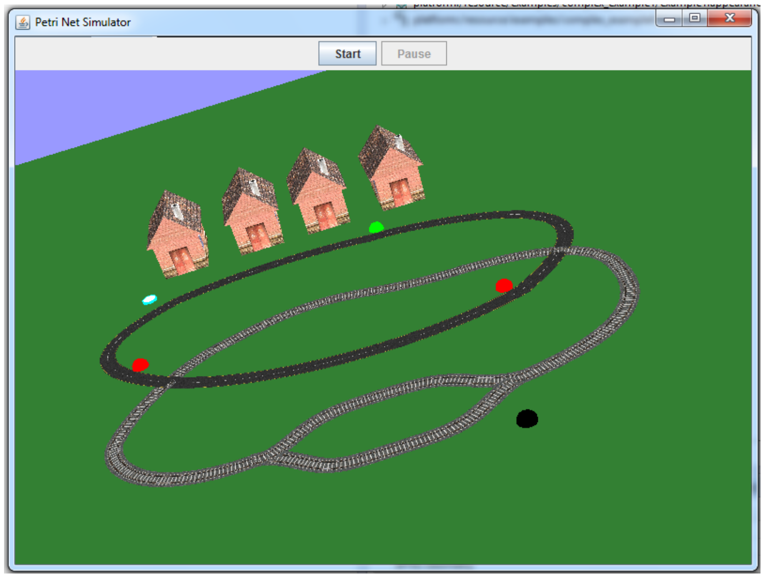


Figure 21: Simulator interface - Before starting

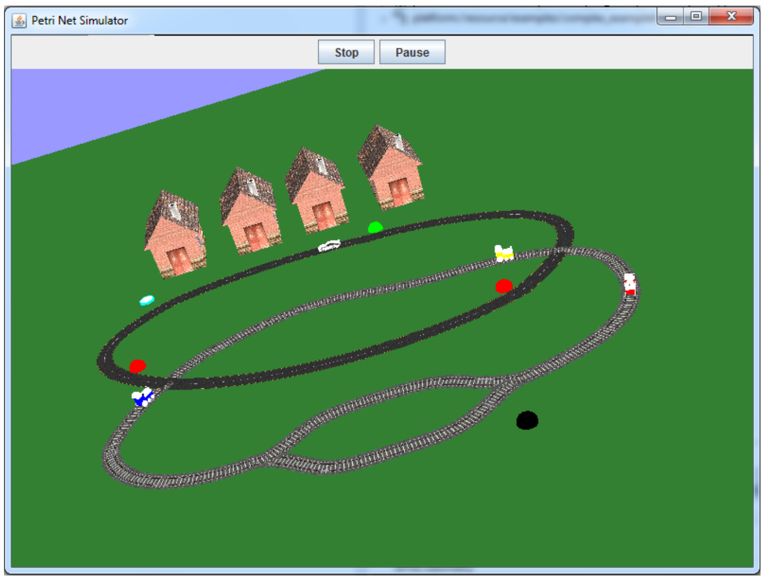


Figure 22: Simulator interface - Simulation running

4 User guide

4.1 Before starting

First, to create a ePNS file, the Eclipse runtime workbench should already be started. In it, the user should create an empty project so that it is later possible to group together all related files (PNML document, Geometry file, etc). An empty project can be created by selecting "File > New > Project > General > Project". Once this is done, there should be an empty folder in the Project explorer view.

4.2 Petri net editor

Author: ***Juan***

4.2.1 Creating a new ePNS Petri net

The creation of a Petri net file in an Eclipse runtime workbench will be detailed in this section. In this file the user will create the Petri net he/she would like to simulate and visualize.

Once the project has been created, the user can create a new Petri net file. To do so, he must right click on the project folder in the Project explorer view and select "New > Other > ePNK > PNML Document" or do this via the "File" menu. The file creation wizard should have opened in a new window in Eclipse where a parent folder and a name for the Petri net file can be selected. The parent folder for the file can be chosen from the existing folders in the Eclipse environment (it is here one can select the empty project previously created to start grouping together all the files for the simulation). Any name can be given to the file as long as the *.pnml* extension is kept. Once these two options have been adjusted, the file can be created by clicking on the "Finish" button.

Now that the file is created, some initial changes should be made in order to allow the edition of the Petri net. First, an ePNS Petri net document has to be created in the file. To do this, the recently created .pnml file should be opened and a child should be added to the "Petri Net Doc" by selecting the "New Child > Petri net (ePNS Petri net)" option in its right-click pop-up menu. Once the ePNS Petri net is created, a page should be added to it as a child by right clicking on it ("PetriNet") and selecting "New Child > Page".

One can now begin the edition of the Petri net (as described in the following sections). Once the editing process is completed, the Petri net has to be validated to check any errors that may have come up during said edition (e.g. conflicts with Arc Identies). This can be done by right-clicking on the root element in the tree editor and selecting the "Validate" option.

4.2.2 Adding items to the Petri net: Places, Transitions, Arcs, Tokens

This section describes the addition of elements to a Petri net page previously created (see 4.2.1). These elements are the ones that describe the Petri net and it is from them that the simulation is inferred. This whole creation process can be done through a graphical editor that is shown by double clicking on the previously created page (see 4.2.1).

Three types of elements (nodes, arcs and labels, where tokens are a specific case of the latter) can be added to the Petri net, each of them in a different way.

- The nodes of the Petri net can be either places or transitions, but they are all inserted in the same way. In the graphical editor the desired component must be selected by clicking on its corresponding icon in the *Palette* shown on the right-hand side of the screen. After this, clicking on the canvas will insert a component of the selected type.
- The arcs of the Petri net connect nodes in the Petri net[10]. This can be done in the graphical editor there is an icon for arcs in the aforementioned Palette. Selecting it will enable the user to input arcs, which can be done by clicking on a node (this will be the arc's source) and dragging the mouse over to another node (this will be the arc's target).
- The tokens of the Petri net are added to the places that have been previously created. The graphical editor presents the tokens as labels: to add a token to a place, create a label on the canvas (this can be done by clicking on the Label icon in the *Palette* and clicking on the canvas) and link it to the place as a token (linking can be done by clicking on the outbound arrow that the label presents will selected and dragging it on to the place or by clicking on the Link Label icon in the *Palette* and clicking on the label and the place; the label will represent a token if the *Token* option is selected in the pop-up menu that appears when linking the label to the place).

4.2.3 Setting some characteristics

Once the components of the desired Petri net have been added to the page, some editing is in order to customize it. This editing is done through the Properties view (shown when clicking on an element and selecting the "Show Properties View" in its right-click pop-up menu) or through the graphical editor, depending on the charasteristic to set. The different attributes that can be changed for each element shall now be described:

Places have main four attributes that can be modified: *Id*, *interactiveInput*, *Geometry Label* and *Animation Label.* The Id is used to identify the place; interactiveInput indicates if the place is an inputPlace, i.e. a place into which tokens can be inserted from the simulation, through a mouse click; the Geometry Label is used to associate the place with a geometry element (See 4.3.2); and the Animations label contains the animations associated to the place (See 4.2.4). The Id and interactiveInput feature can be edited in the Properties View by selecting each one and modifying its value (either writing on the text box or selecting from the options provided, depending on each attribute). The Geometry and Animations Labels must be added through the graphical editor by selecting a *Label* in the Palette and clicking on the canvas. To associate it to a specific place, the user can either click on the label and drag the outgoing arrow that will appear over to the place, or click on the Link Label icon in the *Palette* and click then on the label and the place. When releasing the click button, the *Geometry Label* or *Animation Label* option should be selected from the pop-up menu that is shown. The text in

[10]Please note that arcs can only connect two different types of nodes between them.

the label can be edited by selecting the label and writing the desired animation (See section 4.2.4). Besides, if the place is an Input Place, a fifth attribute can be added: Input Place Appearance Label, a Label that indicates the appearance of the tokens externally inserted in that place. This label is created in the same way as Geometry or Animation Labels.

Tokens in Places can have an *appearance* associated to them, which indicates their graphical representation in the simulation. This characteristic is set by editing the text of the Token Label (this text can be anything, although later it must coincide with the name of an appearance created in the Appearance Editor; for more information see 4.4).

Transitions have the attribute *Id*, which can be modified in the Properties View.

Arcs have one attribute, *identity*, which can be edited through the Properties View. It is used to determine the trajectory of the tokens when a transition is fired.

4.2.4 Animations

Introducing animations into a Petri net model will allow the user to define the way in which the behaviour of the Petri net is represented in the final graphical visualization. The animations will be applied to Tokens when they move into a Place, so it will be in these last elements where Animations shall be edited. This is done by writing on an *Animation Label* in the Petri net editor (see 4.2.3 for more details). Here the different types of Animations and their parameters shall be described, as well as the syntax that must be used to properly insert them into the Petri net.

The animations that can be used for the visual representation of a Petri net are the following:

Move: written as *move(speed)*, this Animation will move graphical representation of the Token (see 4.2.3) at a certain *speed* along the track assigned (as a Geometry Label) to the Place.

Wait: written as *wait(time)*, the 3D Engine managing the visualization will simply do nothing to the Token's representation for the specified amount of *time* (in miliseconds).

Show: written as *show(label)*, it shows the object referenced by the *label* (see 4.4 in the position of the 2D plane connected (as a Geometry Label) to the Place.

Hide: written as *hide()*, it hides from view the object that is in the position of the 2D plane connected (as a Geometry Label) to the Place.

Sequence: this is a special type of Animation composed of two or more of the previously described Animations. It must be written following the scheme *animation;animation;...*, i.e., as animations separated by semicolons.

4.3 Geometry Editor

Author: ***George, Jerome***

4.3.1 Creating a new Geometry file

This section will explain how a Geometry file can be created using the Eclipse runtime workbench.

A Geometry file can be created with the use of wizards. Like all Eclipse creation wizards, the one that is needed for Geometry file creation can be accessed via the *New* menu, either by clicking on the "File" menu of the menu bar or by right-clicking on the Explorer of the workbench. The user then has to choose "Other..." for moving to the next "Select a wizard" dialog. What has been done so far could be also achieved by pressing "Ctrl+N", a shortcut that makes things simpler. Now, the user is ready to finally choose the type of file he wants to create. In the "Select a wizard" dialog he can either type the string "Geometry Diagram" - note that is not case sensitive - or he can find this option under "Examples", in the same dialog.

In the dialog "Create Geometry Diagram" the user can change the name given to the graphical editor. In the last step, by selecting "Next" he can optionally change the name of the tree editor as well. This task is completed by clicking on "Finish". Now the user should be able to see the new project place in the runtime workbench Explorer.

4.3.2 Geometry elements: TrackPosition, Track, SimplePosition

There are two editors with which the user can create a Geometry, the tree editor and the graphical editor. Nevertheless we will see later that the graphical editor is safer and easier to use, so the tree editor shouln't be used in most of the cases. These editors can be populated with some elements, namely *Tracks*, *TrackPosition* and *SimplePosition*.

A **Track** represents the path that a track-bound object will move on. A **TrackPosition** denotes either the starting point of a Track or its ending point. As as far as the Track element is concerned, it should be linked with two TrackPosition so that the Geometry is valid. The two following sections (4.3.3 and 4.3.4) will explain how to add such objects, describing the editors in details. Last but not least, a **SimplePosition** represents a point on the Geometry of a particular interest. On this point a static object, a traffic light for instance, can be put to control the behavior of the track bound moving object that follows the path that a Track elements has defined. A SimplePosition can be placed anywhere in the Geometry.

4.3.3 The tree editor

The tree editor is the one of the two editors that becomes available after the creation of a new Geometry file. The file-format for this editor is recognized by the ".geometry" suffix. Double-clicking on a file of that type will start the tree editor. The user can also edit this type of file by clicking on the little arrow on the left of the filename ending with a ".geometry" while he is inside the Explorer. With this editor the user can create and delete elements of the Geometry. However it is advisable to use the graphical editor for that purpose instead, since it is easier to use and the user gets an immediate picture of the changes he/she is doing to the Geometry. Indeed, the use of the graphical editor for creation and deletion of Geometry elements is safer, since inconsistencies might emerge otherwise between the two editors. This is the reason why this document does not elaborate more on how to create or delete Geometry elements in the tree editor. Figure 23 shows an overview of the tree editor editing a Geometry file.

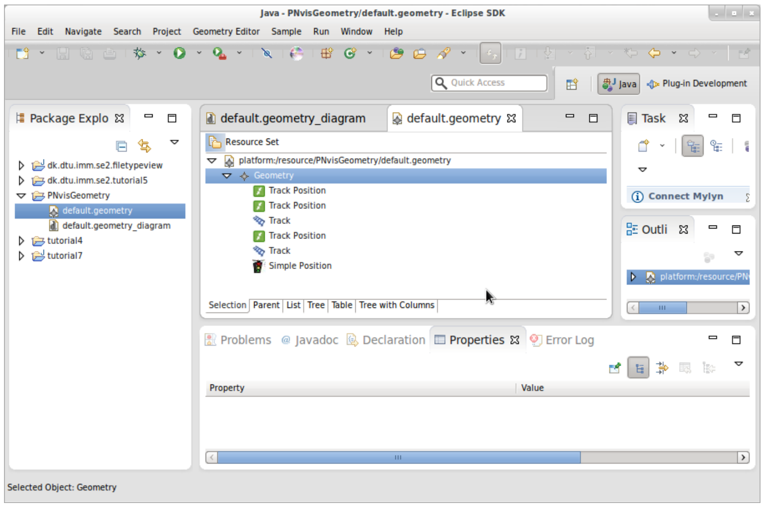


Figure 23: Overview of the tree editor for a Geometry file

An important action that the user can perform using the tree editor is to validate the Geometry he has built. This can be done by right-clicking on the top element of the tree editor. Then, in the pop-up menu he has to select "Validate". If the validation succeeds the user will be notified with a message on the emerged dialog. In case of validation failure the same dialog will give the user the user the chance to investigate failure related messages by clicking on the "Details" button.

4.3.4 The graphical editor

The graphical editor is the second editor that becomes available after the creation of a new Geometry file. It should be used to edit these Geometry files as it is safer and easier to use than the tree editor. The file-format for this second editor can be found thanks to the ".geometry_diagram" suffix. To open this editor the easiest way is to double-click on a file with that suffix and that previously created inside the Explorer (see section 4.3.1). Figure 24 shows an overview of the graphical editor with an open Geometry file.

On the right-side of the editor, the *palette* or tool bar of the graphical editor can be seen. This allows the user to create Geometry objects. We will come to that part a little bit later. At the top, the *tab* of this page can also be seen, displaying the name of the document.

To create Geometry objects the user needs to click on the element he wants to create inside the *palette* and drop it where he wants it to be inside the canvas. There are three different objects he can create with the graphical editor for Geometry files : a TrackPosition, a Track, and a SimplePosition. He can have a look at section 4.3.2 to understand in which case each element should be used. It must be noted that the reader cannot create a Track if he/she has not previously created at least one TrackPositions.

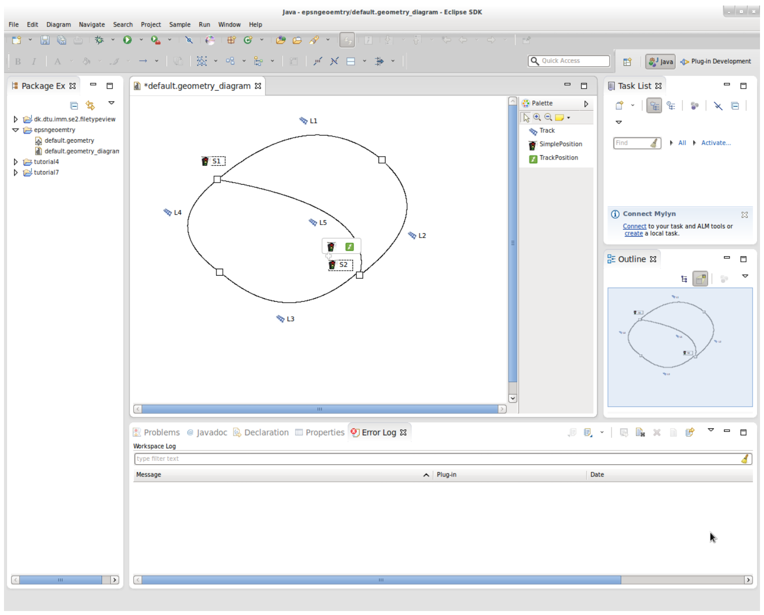


Figure 24: Overview of the graphical editor for a Geometry file

4.3.5 Setting some characteristics

Ultimetaly, the reader must set some characteristics of the Geometry objects in order to make the simulation running with appropriate visuals. These characteristics can be inspected at the Properties View for each object. If this view is not visible in the workbench, go to the menu "Windows" -> "Show View" -> "Other..." and inside "General" double-click on the "Properties" (view). This view will now display additional properties for each object selected. Some of this properties are already defined, such as the starting point and ending point of a Track, while others need to be defined, for instance the AppearanceLabel of a Track.

To configure correctly the Geometry file the user must define some of these additional properties. For the Track object and the SimplePosition object the user will have to define an AppearanceLabel that is a label refering to some appearance defined in the Appearance Editor (see section **??**). This can be done by clicking on the object once created and double-clicking in the right-zone corresponding to the AppearanceLabel inside the properties view, then writing some text using the keyboard. For the TrackPosition object the user will not have to define anything more.

Finally, the user can change the Label of some Tracks and SimplePositions by clicking on the name-label inside the graphical editor and start typing the new name or by selecting the object and changing the value of the field Label inside the properties view.

4.4 Appearance Editor

Author: ***Pablo***

4.4.1 Creating a new Appearance file

The creation of an Appearance file is done by selecting "New > Other > Example EMF Model Creation Wizards > Appearance Model" and clicking on the "Next" button. The next window allows both the selection the parent folder for the file and the edition of the name of said file (maintaining the *.appearance* extension). After editing these two things and clicking on the "Next" button, the option "ePNS Appearance Model" must be chosen as a Model Object. Clicking on "Finish" will create the Appearance file.

4.4.2 Adding items to the Appearance: Model3D, Shape3D, Texture, SurfaceColor

Once there is an Appearance file created (see above), elements can be added to it. To do it the Appearance file must be open and its root node expanded. In said node will be a node "ePNS Appearance Model", where right clicking permits the addition of an element to the Appearance file by means of the "New Child" option. Here the user can choose between four elements: Model3D, Shape3D, Texture and SurfaceColor.

4.4.3 Setting some characteristics

Simply having an Appearance file created and adding elements to it will not suffice, for these elements need to refer to something that describes the Appearance, as well as having a name to associate them to the components of the Petri net. The attributes of each element that can be edited are here presented, each of them editable in the Properties view of the selected element:

- All elements have a *label* attribute that gives each one a name and will later allow the connection to the components of the Petri net.
- Both **Model3D** and **Shape3D** objects have: a *scale* to make them bigger or smaller; three rotation values (*xRotation*, *yRotation* and *zRotation*) to rotate them; and an *elevation* attribute to raise the shape.
- **Model3D** also has a *file* attribute, which is a reference to a file where the desired Appearance is described. This file must be located in the Ecplipse workbench.
- A **Shape3D** also allows the modification of its *type* attribute. This attribute grants the user the choice to select the object's Appearance among a series of predefined shapes (cube, sphere...).
- A **Texture** has a *file* attribute analogous to the *file* attribute in the Model3D (again, this file must be located in the Ecplipse workbench).
- Finally, the **SurfaceColor** will have a *Color*, which can be chosen from the 13 predefined colours.

Shapes (Model3D and Shape3D elements) can also have a surface (Texture or SurfaceColor). To add this, right-click on the desired shape and select "New Child" and "Texture" or "SurfaceColor". The properties of the selected surface can be changed again using the Properties View.

4.4.4 Adding appearances to Tokens, Tracks and Simple Positions

To add appearances to Tokens, Tracks and SimplePositions, one must have created both a Petri net and a corresponing Geometry. See 4.3.5 and 4.2.3 for information on how to do this.

4.5 Configuration Editor

Author: ***Marius***

4.5.1 Creating a new Configuration file

This subsection gives details on the creation and proper setting of a Configuration file. With all the Geometry, Appearance and PNML resources created, a Configuration file is needed, to link them all together. The Configurator gives the user the possibility to use, for example, the same Geometry and PNML files, but with a different Appearance file. To create a new Configurator file, the user must go to File -> New -> Other, and from Example EMF Model Creation Wizards select Configurator model and click Next. From there, he should select a parent folder, give the new file a name and click on Finish.

4.5.2 Loading resources and setting Configurator properties

To load the resources and set the Configurator properties, the user should first open the Configurator file. In the tree-view, it can be seen that the file contains a Configurator object. Here, the Appearance, Geometry and PNML resources need to be loaded. This is done by right-clicking in the active window and selecting "Load Resource". Then, after having selected "Browse Workspace", the Appearance file created as in step 4.5.1 must be selected and click Ok. The same process must be repeated for the Geometry and PNML files. The order in which these files are loaded is not important. They can also be loaded all at once.

To set the configurator properties, the user has to click on the Configurator and in the property view, in the Appearance field, he should click on the Value field and select "ePNS Appearance Model" from the dropdown menu. Similarly, for the Geometry property, "epns.geometry" must be loaded, and for Petri net, "Petri Net Doc" must be selected. Another property that the user should set is the Default Track Width. Its default value is 1.0, but a bigger value is recommended, so that the tracks can be easily distinguished.

4.5.3 Starting a simulation

With the Geometry, Appearance and PNML resources loaded and selected in the configurator, to start the simulator, the user has to right-click on the Configurator object and select the *Start simulator* menu item. A new window will appear, in which the simulation would run. More details can be found in section 4.6.

4.6 Simulator

Author: ***Ruxandra***

4.6.1 Starting the simulator

The Simulator can be started by clicking on the Start Simulator option after right clicking on the Configuration item from the .configurator file, after loading and linking the geometry, the appearance and the Petri net files, as described in 4.5.

4.6.2 Controlling the simulation: Play/pause/stop

After starting the simulator, a new window like the one shown in figure 25 will be displayed. The simulation can be played by pressing the Start Button on the left side of the window.

The simulation can be stopped at any time. By stopping it, the simulation will return to the initial state. The simulation can then be played again by pressing the Start button.

At any time, the simulation can be paused, by pressing the Pause button. It can then be resumed by pressing the Resume button, which will replace the Pause button after a pause.

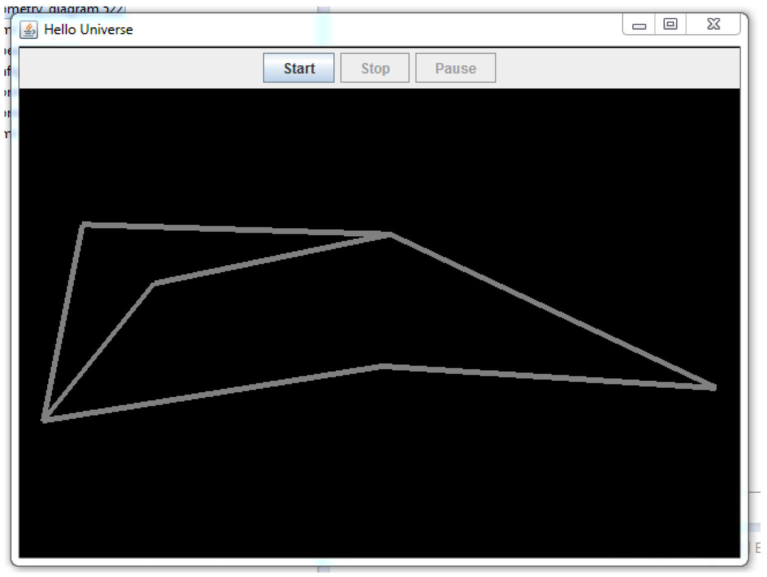


Figure 25: Simulator interface

4.6.3 User Interaction

The user can interact with the animation by using the input places. The interactivity for a place is set in the pnml file as described in section 4.2.3. If a place is an input place, it usually has associated an appearance as a Single Object and is placed on a SimplePosition (see section 4.3.2 for details) , rather than a track. In this way, the user can click on the object visually associated to the place and insert a token into the place. Another way the user can interact with the simulation is by changing the view point. There are 3 ways to move that camera: scroll to zoom in/out, right click and drag to pan the camera and left click and drag to rotate the camera.

5 Glossary

Author: ***Georgios***

5.1 Technical Glossary

Animation The dynamic behavior of objects on 3D Canvas that are associated with Places of the Petri Net model.

Appearance The look and feel of the objects visualized by the 3DEngine .

Appearance Editor A graphical editor provided EMF for configuring appearance.

Control GUI A simple Graphical User Interface with appropriate actions (play/pause/stop buttons) so as the end user to be able to interact with the 3DEngine and consequently with underlying Petri Net execution .

Geometry the track (3D line/curve) on which the 3D-visualization of the Petri Net simulation takes place. The geometry consists of predefined objects such as, semicircles, points and lines.

Geometry Editor A graphical editor provided by ePNK and GMF for designing the area on which the simulation will be reflected.

Identity The attribute of Arcs in the Petri Net model that defines the trajectory of the Tokens.

IgnoreAnimation The attribute of Arcs in the Petri Net model that allows the target transition to fire without the associated Place Animation having finished.

Input place a place in which tokens can be insterted externally, i.e. by clicking an interactive control point.

Interactive control point A point that allows the user to interact with the system (e.g. by a click)

Petri Net Editor A graphical editor provided by ePNK and GMF for designing the Petri Net of the system a user wants to visualize.

Physical object A graphical object that is used by the 3DEngine for visualizing its behaviour during the Petri Net simulation.

Simulator The software component that will provide the underlying Petri Net execution information to the 3DEngine so as to be able to visualize the Petri Net execution with 3D objects.

Shape The visual traits of a physical object being part of the simulated system visualization.

Token The Petri Net element that moves along Petri Net places through transitions.

5.2 Technology Terms

Ecore Tool editor - The EMF editor that provides all the necessary elements for realizing or drawing the model in question.

EMF - The Eclipse Modeling Framework. It auto-generates code that represents the corresponding model (usually described in UML).

EMF Validation Framework OCL Integration - Helps for expressing constraints on ambiguous graphical models such as a class diagram. Tightly used with UML users can define constraints on their models.

ePNK - The eclipse Petri Net Kernel, a modern equivalent of PNVis missing the visualization part though. Built following the model-based software engineering paradigm. ePNK Petri Net Types to be extended to provide new functionality.

GMF - The Eclipse Graphical Modeling Framework provides the appropriate infrastructure for developing graphical editors based on EMF.

Xtext - Xtext is a framework for development of programming languages and domain specific languages.

Index