UML Diagrams
Extended Petri net Simulator

Group A

October 10, 2013

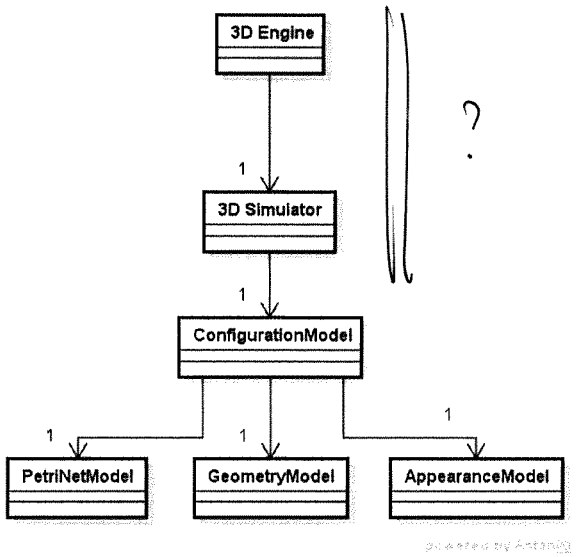
Abstract

This paper contains the UML diagrams for the major components of the Software Engineering 2 project, which it is a Petri net 3D Simulator. More details about the project can be found in the Project Definition document.

Contents

1 Overall model

Figure 1 presents a high level domain model of the entire system. The user interacts with the system using the editors and the simulator.



do not understand the associations here (maybe, this is a component diagram in class diagram disguise??)

Figure 1: Domain model

2 Petri net model

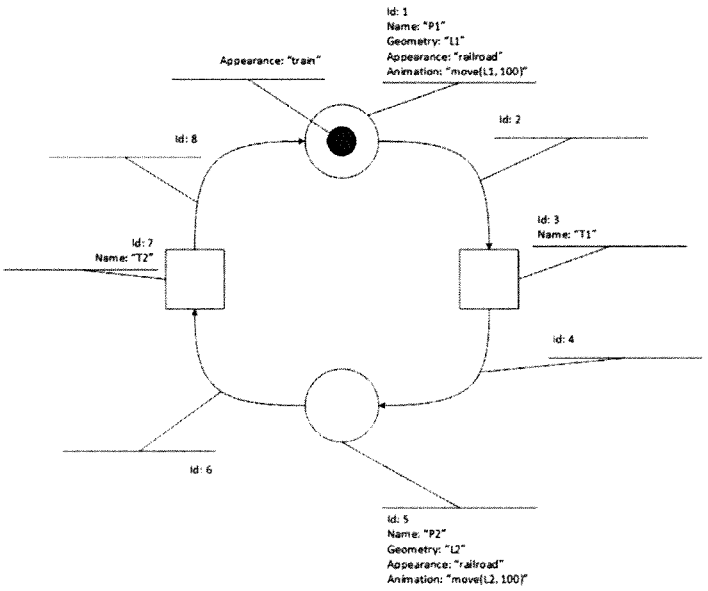


Figure 2: Example of an extended Petri net

Figure 2 shows an example of a particular extended Petri net. On this Petri net, there are two places (P1 and P2), two transitions (T1 and T2), and four arcs that link all the elements. Additionally, places can contain tokens. Moreover, the places contain additional information relevant for the simulator:

- A geometry label, to link the place to a geometry location
- An appearance label, to link the place to a specific appearance, with shape, texture,...
- An input place label, to indicate if the place is a special place that the user can click (for example to change
- the behaviour of the Petri net).
- An animation label, to indicate how the 3D simulator should animate this place and the tokens contained in it.

A token also has an appearance label, in order to define its shape on the simulation.

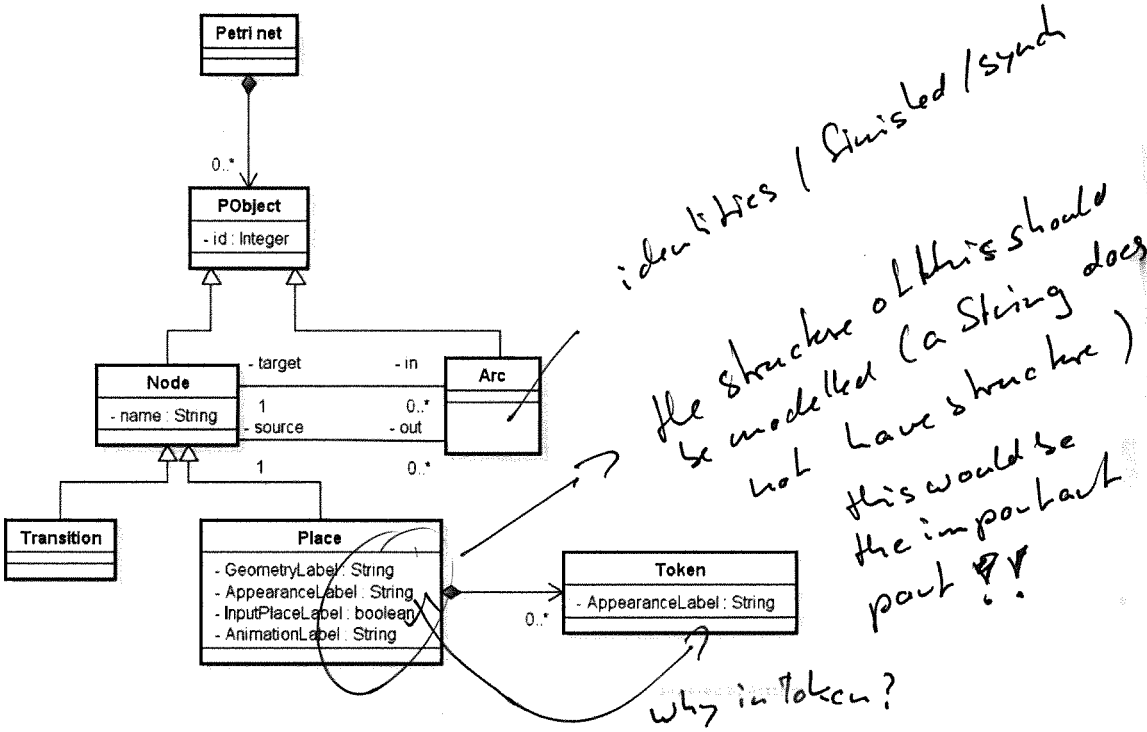


Figure 3: Petri net model

3 Geometry model

The diagram in Figure 4 describes the domain model associated with the geometry editor. The geometry editor can have an infinite number of GObjects, and each one has a name and an id. A GObject can either be a Point or a Line. A Point has a xLocation and an yLocation, which is the location of the Point in a 2D space. Each Line has two connectors, one in each end of the line, and a bendPoint, which allows the user to bend the line to make curves. Connectors can have an infinite number of Lines connected to them. An InputPoint is a point that the user can interact with, to add or remove tokens during the simulation.

Figure 5 is an example of how different lines can be connected in the Geometry Editor. The line with name L1 and id 1 has the two connectors C1 and C2 and the line L2 is also connected to the two connectors, C1 and C2.

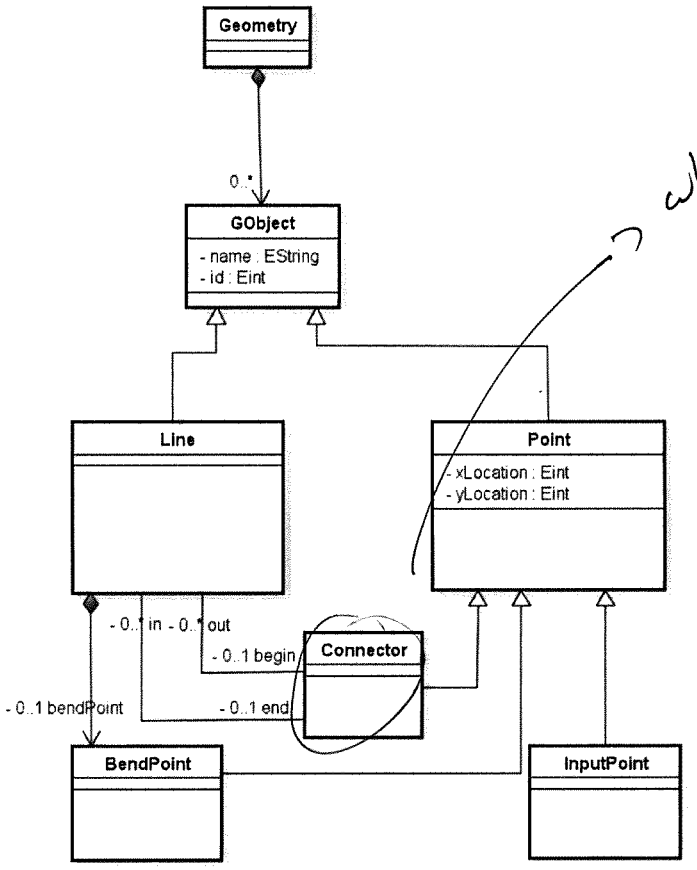
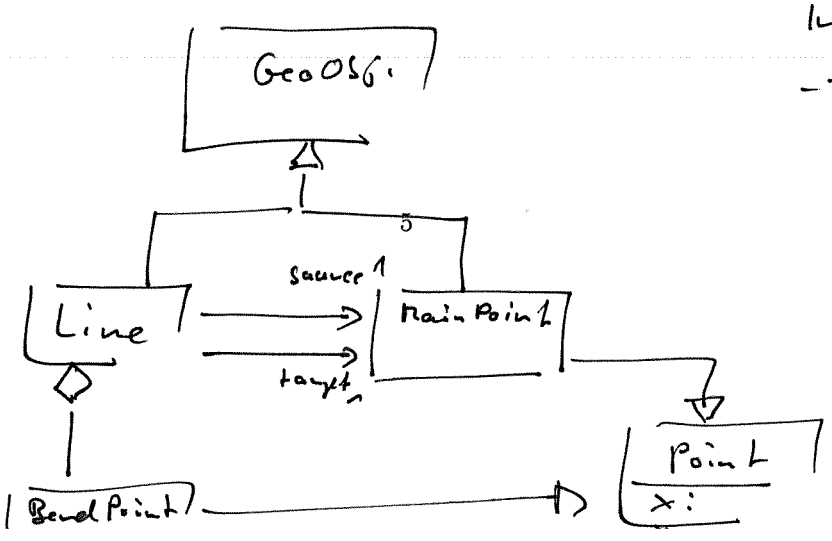


Figure 4: Geometry Model



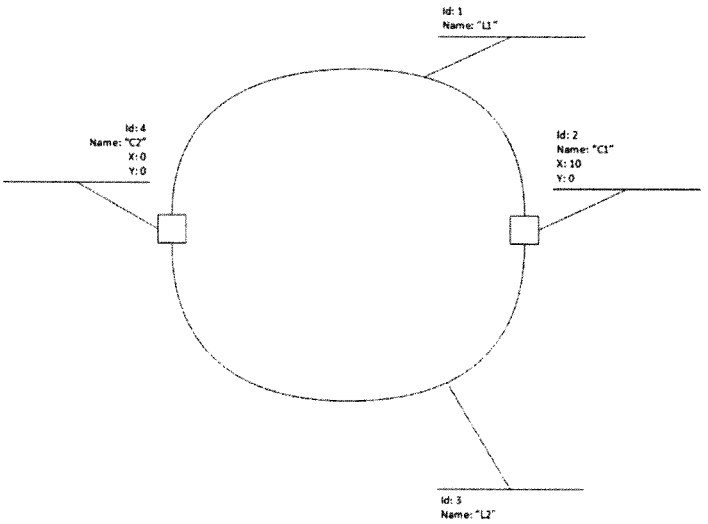


Figure 5: Example of a geometry

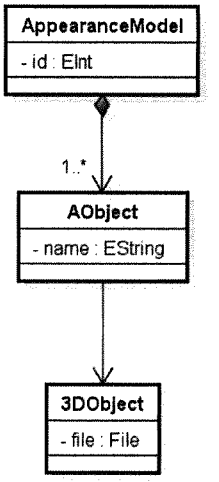
4 Appearance model

The above diagram describes the domain model associated to the Appearance Editor described in the Project Definition. Each appearance model will have one or more objects of type AObject, each corresponding to an element in the Petri Net. The name attribute of each appearance object should be the same as the AppearanceLabel of its corresponding Petri Net object. Moreover, an appearance object has an associated 3DObject which is a file previously defined by the user and loaded in the Appearance Editor. Such a file can contain all or some of the following elements:

- shape: sphere, cube, etc.
- texture
- color: red, green, black, etc.

The Appearance Model will in the end be structured as an XML file which will be used by the Configurator Editor.

↳ but not necessary to explain here (XML)



textures for trades / lines?

→ size of 3D objects (scaling / rotation)

Figure 6: Appearance Model

5 Configuration model

The diagram in Figure 7 describes domain model associated to the Configuration Editor described in the Project Definition. It is quite simple because basically, a configuration model consists of the three files generated by the other editors (Petri net editor, Geometry editor, Appearance editor). As each editor is generating an XML file, the configuration editor takes all these file as inputs. This can be seen in the UML diagram where a Configuration model has an association to each of the other models. This model will be used by the Configuration Editor in order to create the final configuration XML file which will be provided to the simulator.

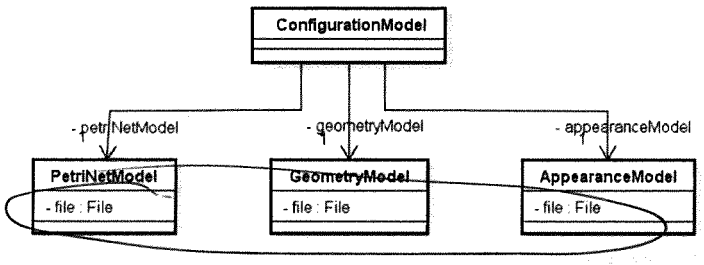


Figure 7: Configuration Model

conceptually and technically you can point from the config to the PM, geometry without referring explicitly to a file ?

6 3D Simulator model

The UML diagram of the Simulator visualizes how the Simulator is dependant on the jMonkey 3D framework, that is used to run the simulation. Also, the Simulator needs a configuration file in order to function. The configuration file is used to specify what the SObjects look like and how they behave; SObjects have an 'id' and a 'name' in order to properly link them with the configuration. The SObjects can be used to create Places that have Tokens. There can be as many Places as desired, and as many Tokens on each Place as desired, but a Token can only have one Place.

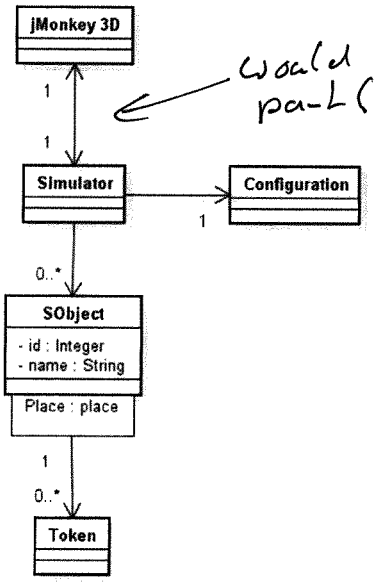


Figure 8: Simulator Model