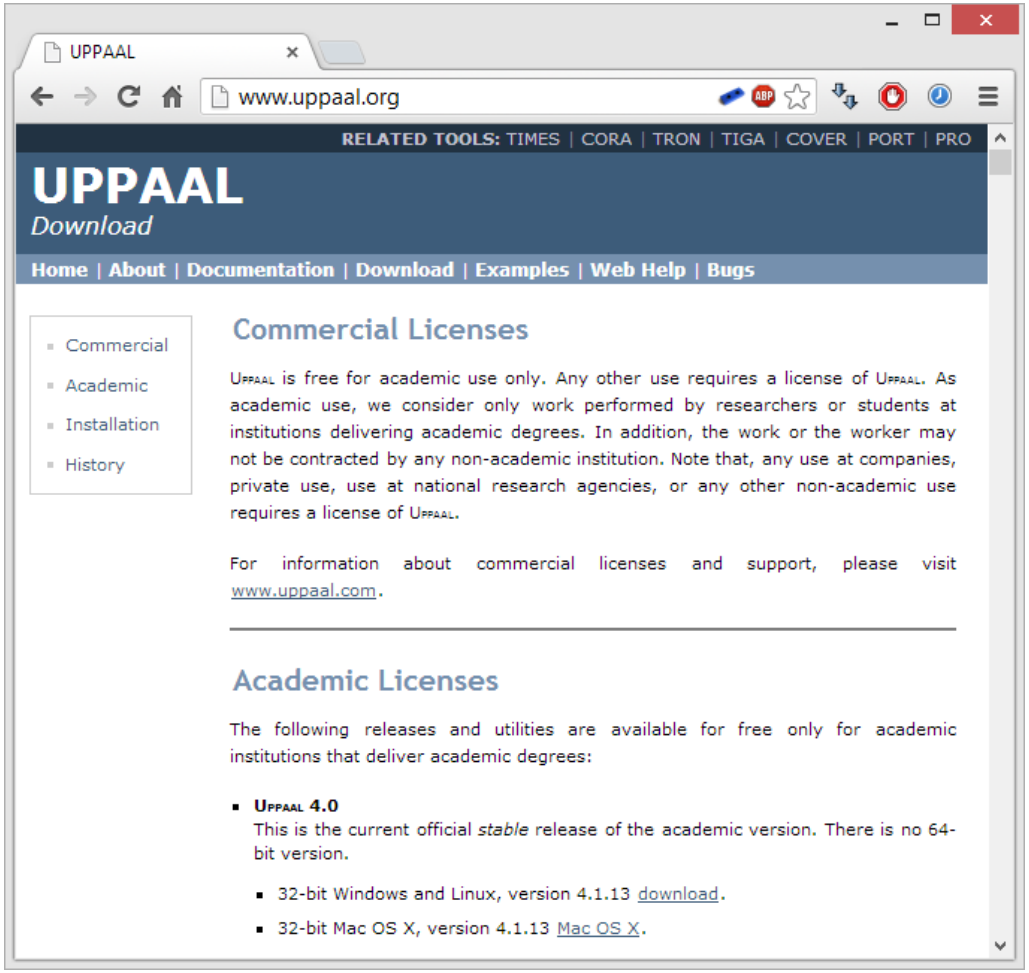
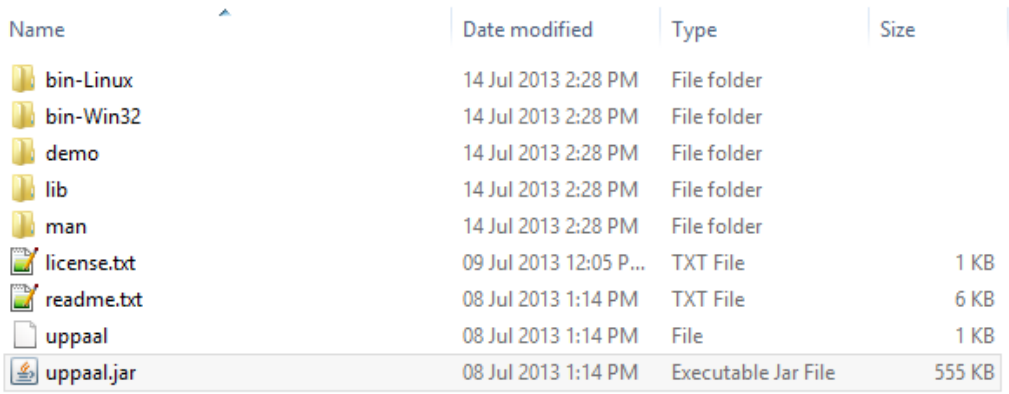
UPPAAL LAB
GIAN 2016

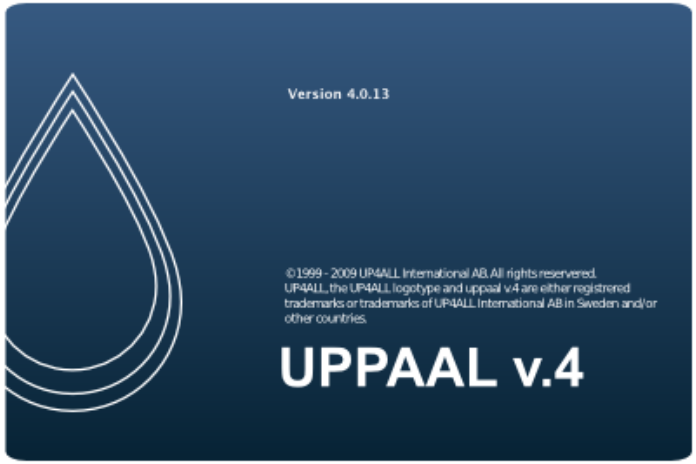
Download UPPAAL 4.0



- Redistribution of UPPAAL is prohibited, so we cannot provide you a copy.
- Download UPPAAL 4.0 from :
  - http://uppaal.org
- You will have to register for download.

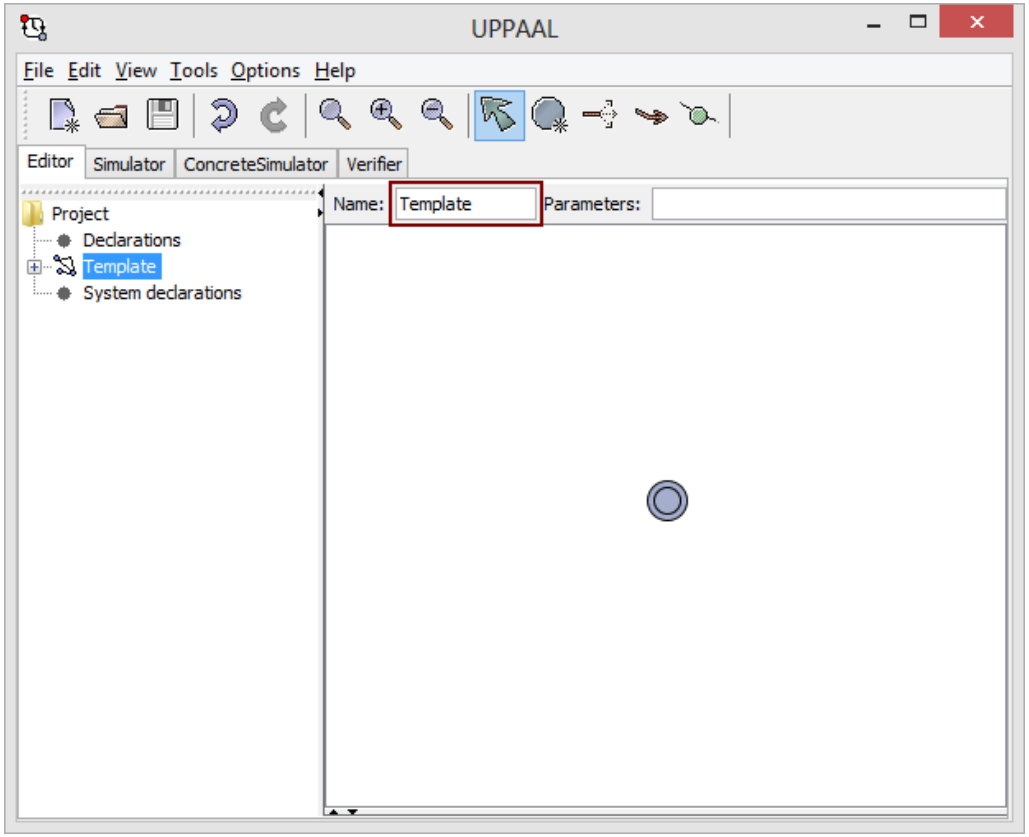
Extract and Run UPPAAL 4.0





- Extract the downloaded archive in a writeable location.
- Double click on uppaal.jar to run the UPPAAL model checker.
- Troubleshooting
  - Ensure that you at least have JRE7
  - If an empty dialog box appears, most probably using the developer release will fix it.
  - If you receive a message “server disconnected”, you need to run the jar with elevated privileges.

First Look



- The UPPAAL model checker has a menu-strip, a tool-box, content-tabs.
- The default content is the component editor with an editing pane and a project explorer.
- UPPAAL has templates for components, that come with a pre-existing initial location.
- You can change the name of the component by editing in the name field.

Train-Gate System

- In this lab, you will create a model of a Train-Gate system.
  - The system consists of a bridge that has to be shared between multiple train tracks. A gate on the bridge allows one train to pass at a time.
  - A train approaching the gate intimates it and attempts to pass through, unless instructed to stop by the gate.
  - Multiple simultaneous request may arrive (Max = 6), which are served on a FIFO basis.
  - While processing a request e.g., letting a train pass through, the gate instructs all other approaching trains to stop until their turn.
  - Upon its turn, a stopped train is instructed to go. The train thus starts again and leaves as soon as possible.

Train-Gate System

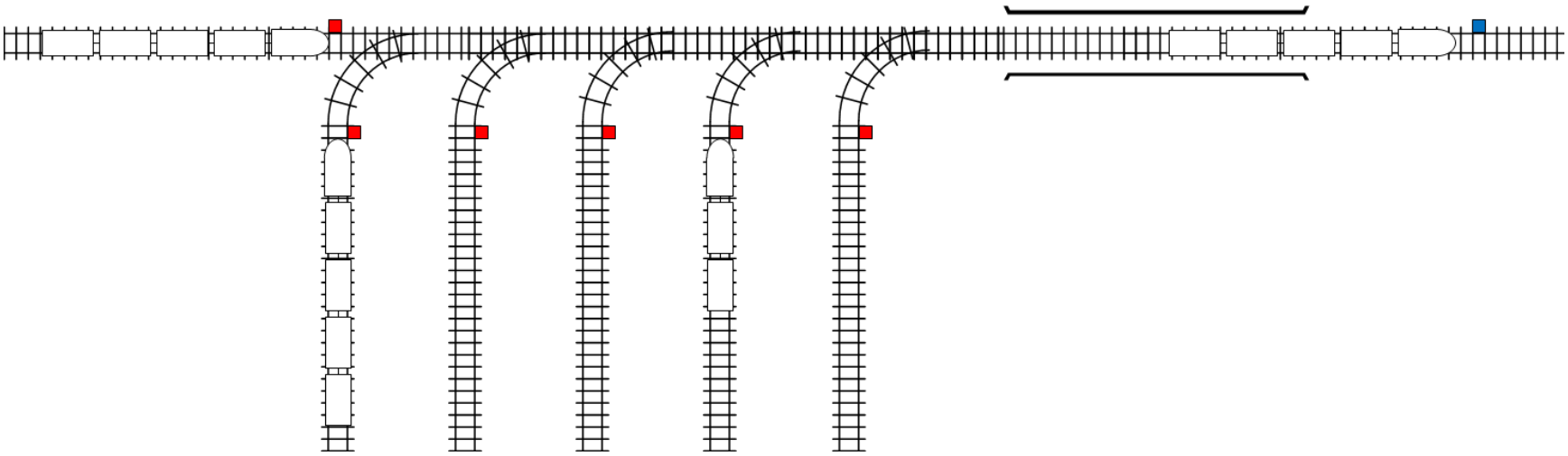
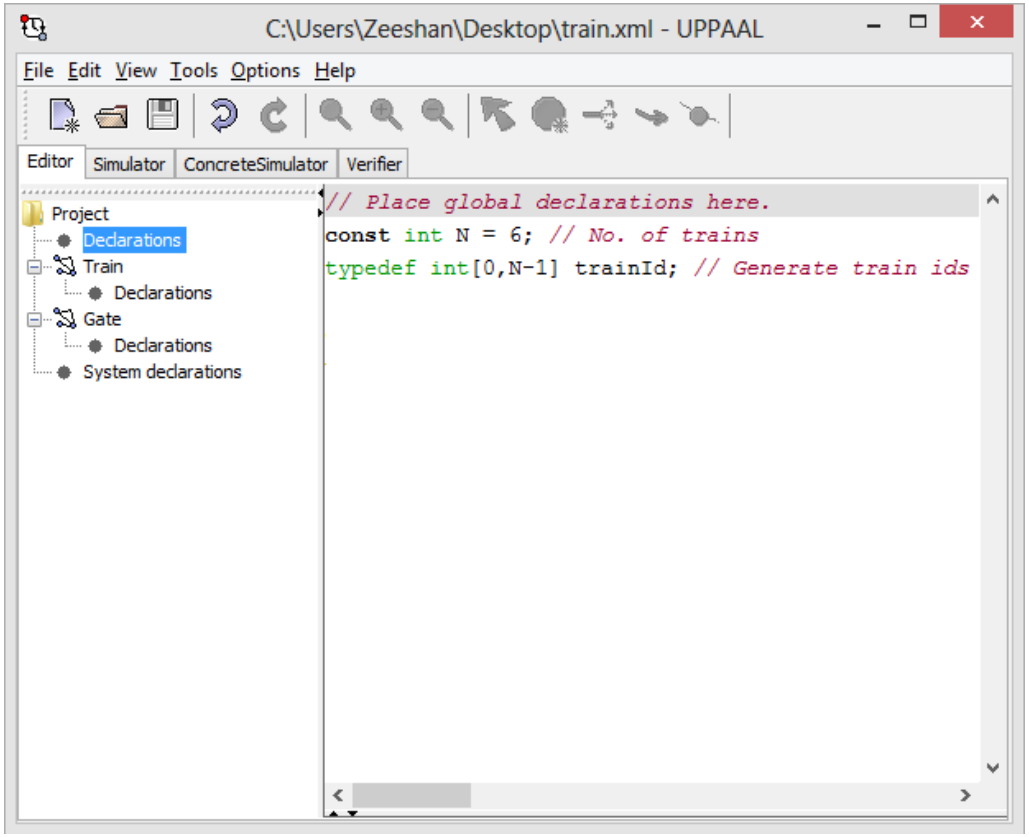


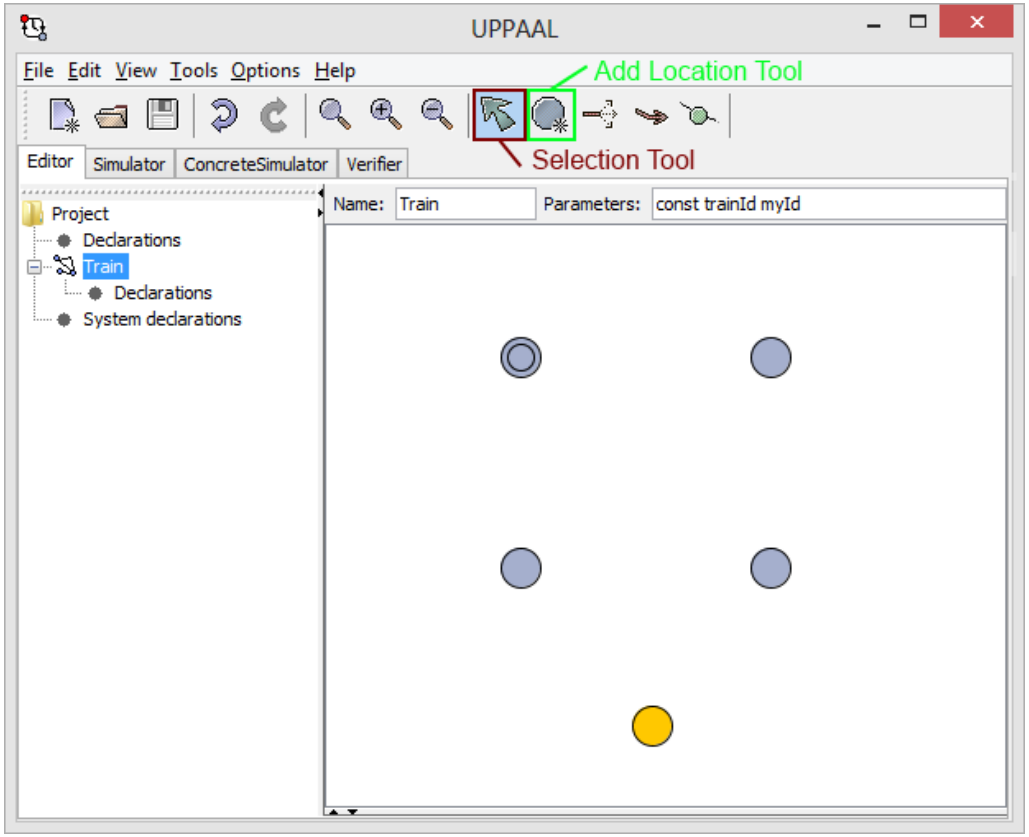
Fig. 1. A railway bridge, shared between multiple tracks. Approach and Leave sensors are visible.

Project Declarations



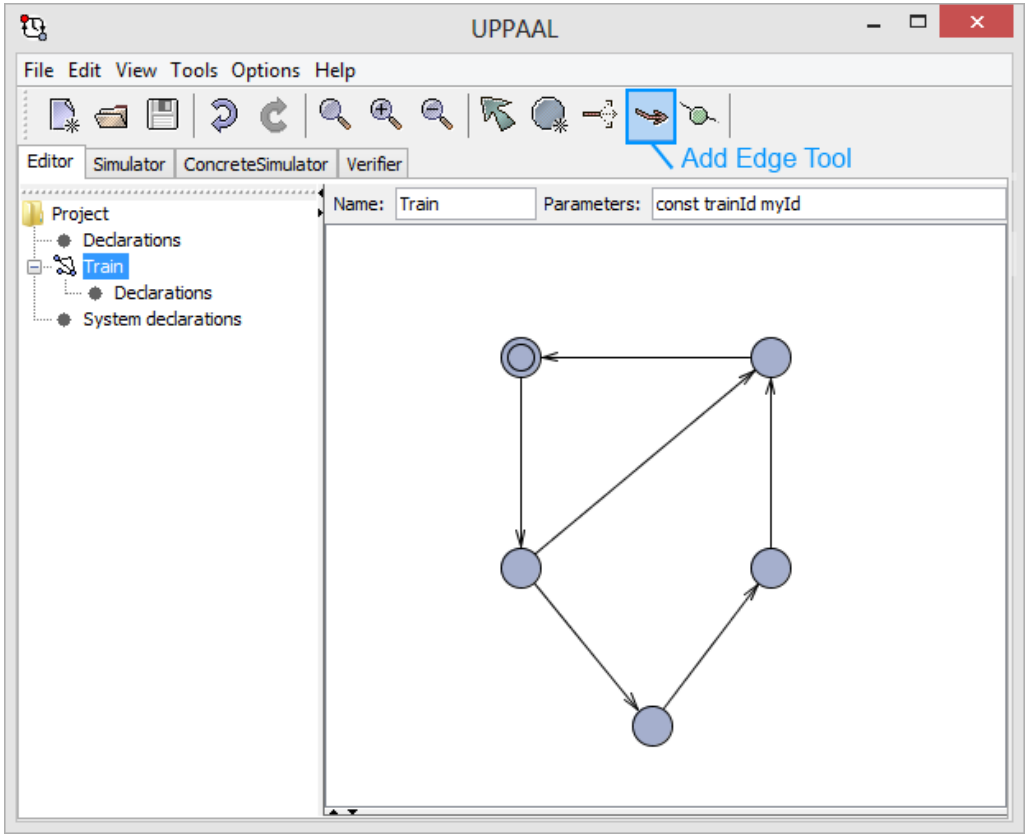
- Select project declarations for creating global variables, constants and channels etc.
- Define a constant integer for max number of simultaneous requests. This mimics 6 trains requesting to pass through.
- Create an enumerated integer type that can vary from “0” to “5”.

Create the Train Component (1/10)



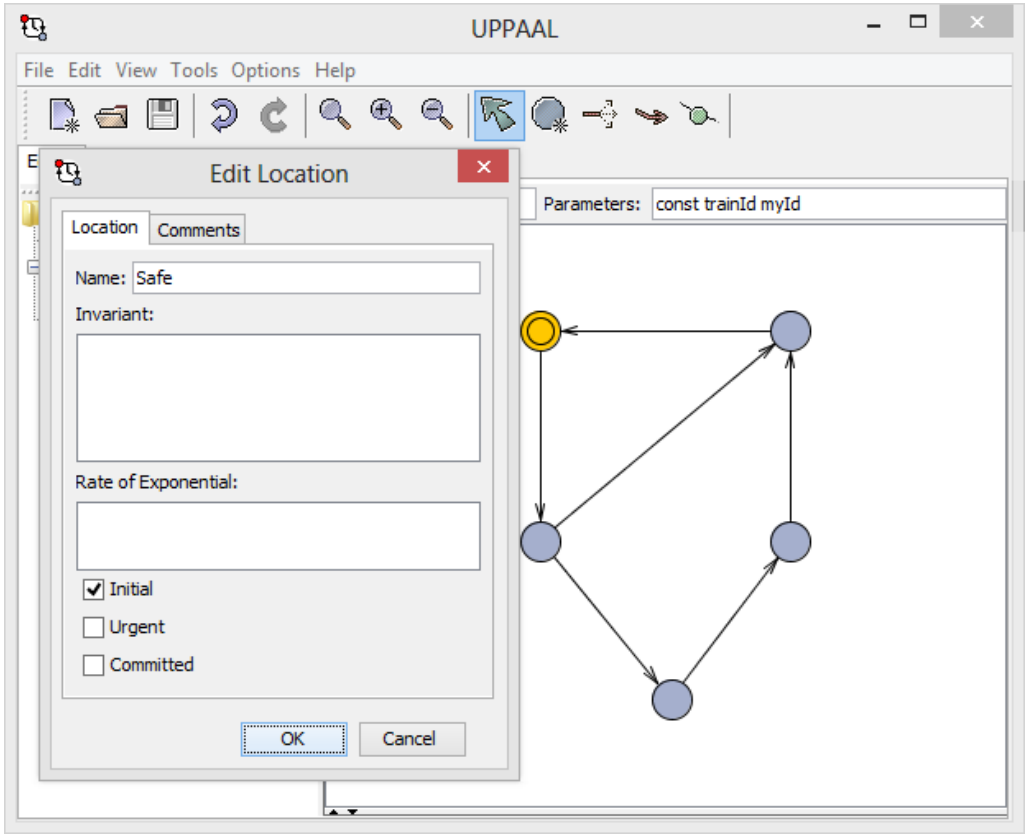
- Rename "Template" to "Train" using the Name field.
- Specify Parameters for this component. This represents that the component may be instantiated multiple times based on the given parameters. For
- Using "Add Location Tool", create 5 locations in your Train component.
- Use the "Selection Tool" to select locations and move them around.

Create the Train Component (2/10)



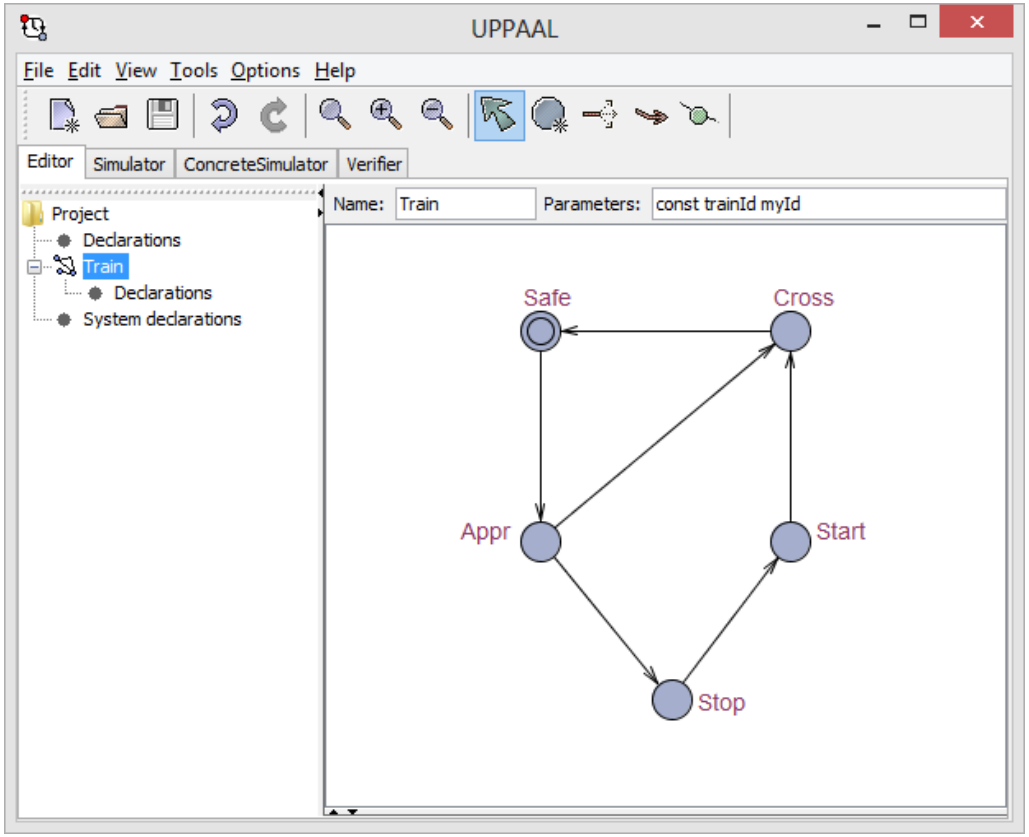
- Using “Add Edge Tool”, create edges as shown in the picture.
- Ensure that the direction of the edges also match.

Create the Train Component (3/10)



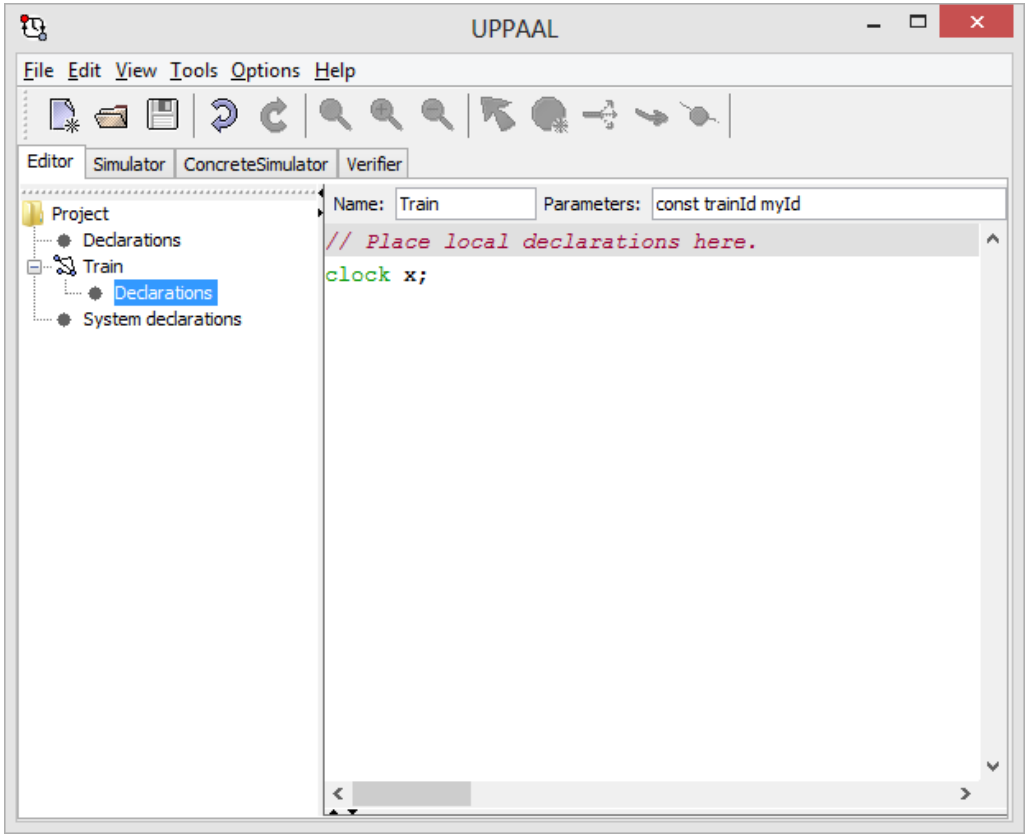
- Using the "Selection Tool", double click on the initial location to bring up the "Edit Location" dialog.
- Specify name "Safe" for the initial location.

Create the Train Component (4/10)



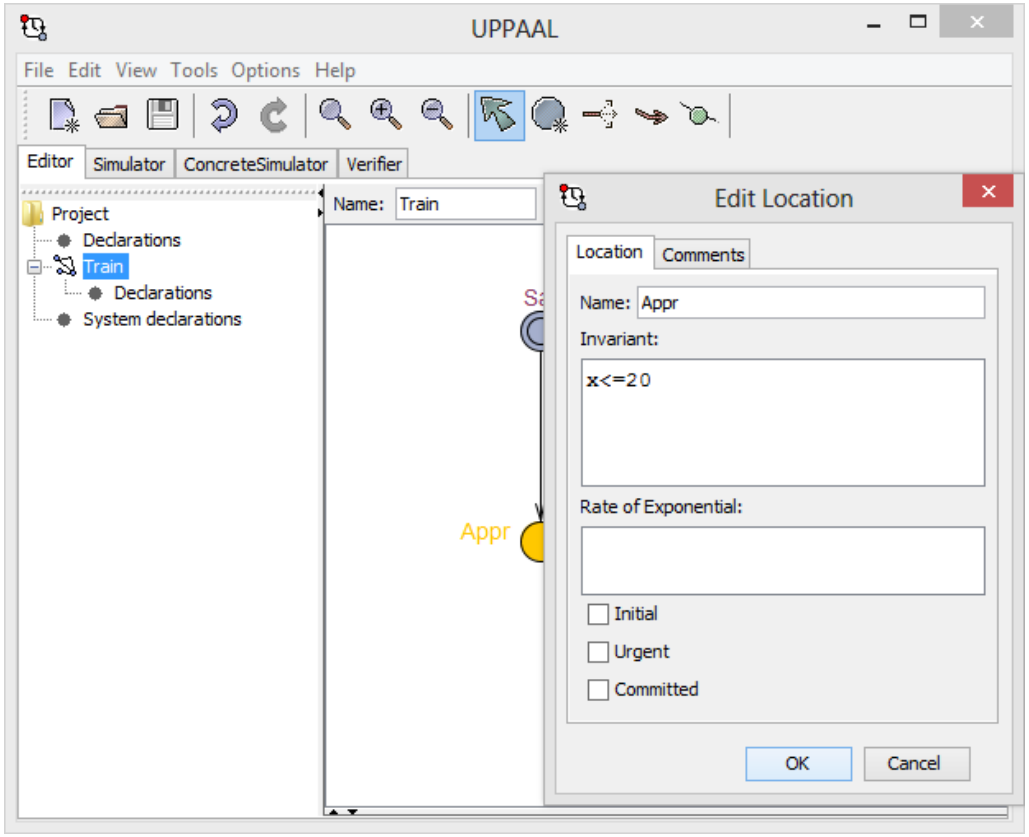
- Similarly, set names for all locations in the Train component, as shown in the picture.

Create the Train Component (5/10)



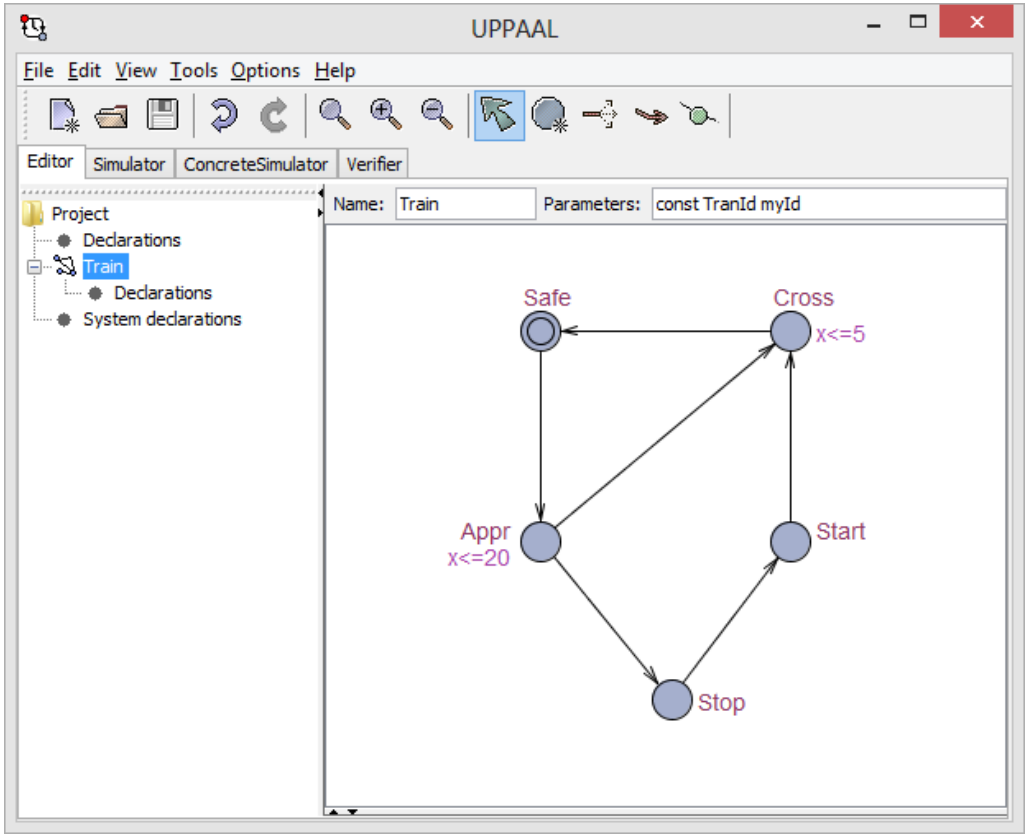
- Expand the “Train” node in the project explorer and select “Declarations”.
- These declarations are local to the “Train” component.
- Declare a clock variable “x”.

Create the Train Component (6/10)



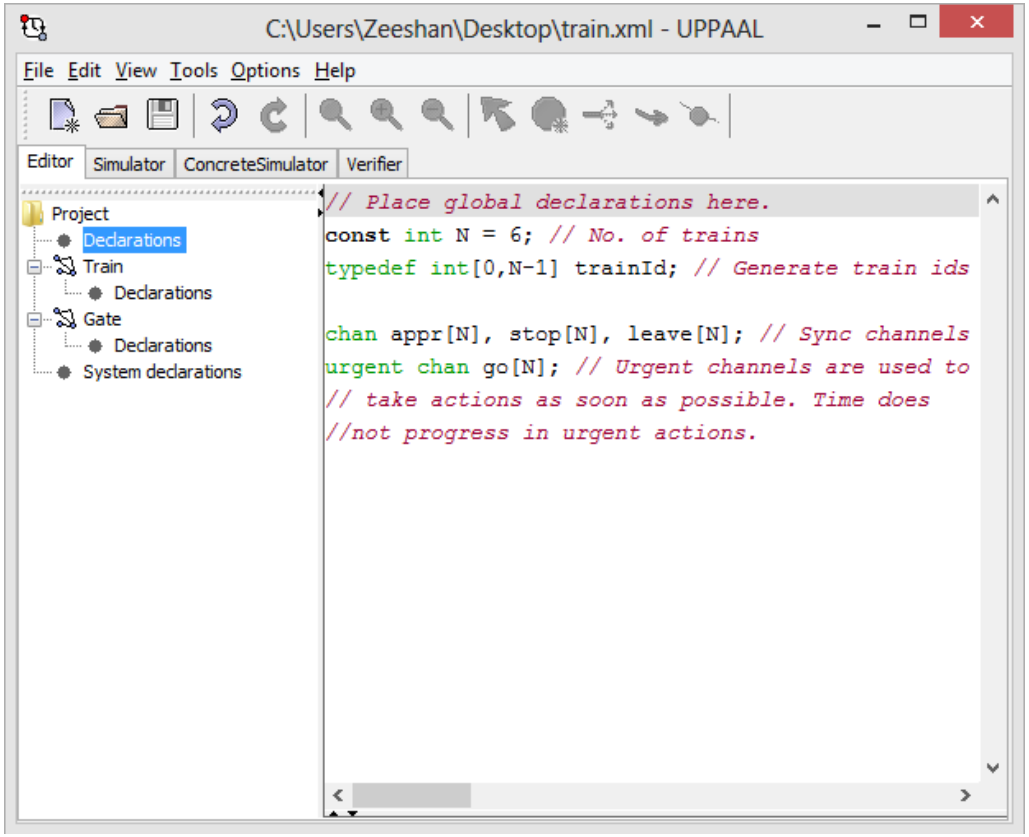
- Select and double click the "Appr" location.
- Specify an condition "x<=20" as an invariant over the clock variable to this location.
- An invariant is a condition should remain true while the system in this location.
- This invariant represents that the train may be in approaching location for up to 20 time units.

Create the Train Component (7/10)



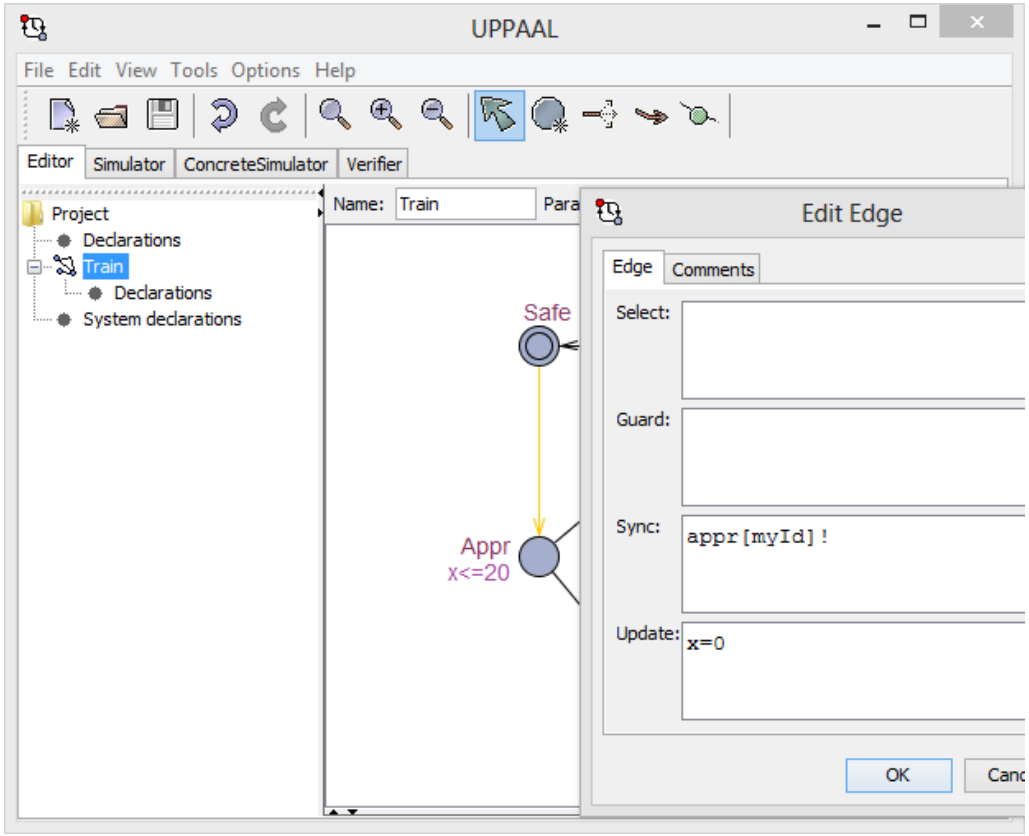
- Similarly create all invariants as shown in the picture.
- Maximum time a train can take to be in approaching location is 20.
- Maximum time a train can take to start is 15 time units.
- Maximum time a train can take to cross is 5.

Declaring Channels



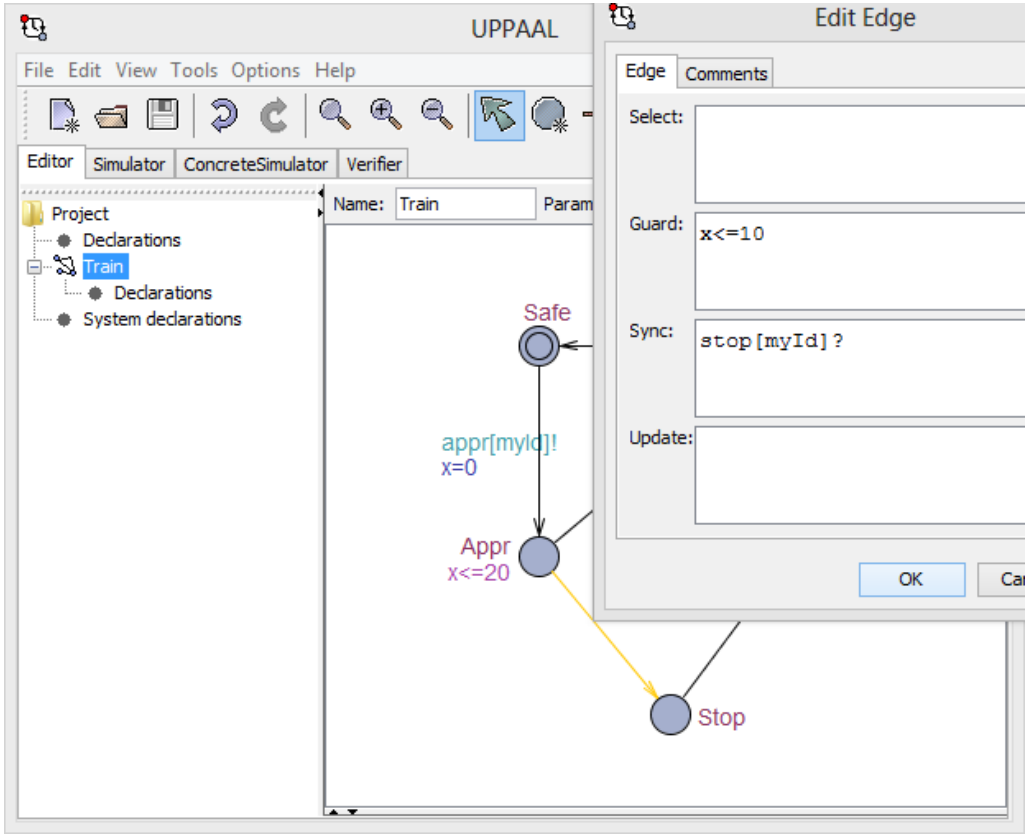
- Channels are used to enable lock-step actions i.e., taking edges together.
- Select project declarations.
- Create 3 ordinary sync channels “appr”, “stop” and “leave” for each train. Using arrays, makes the design scalable.
- Create 1 urgent sync channel for instructing trains to “go” as soon as possible.

Create the Train Component (8/10)



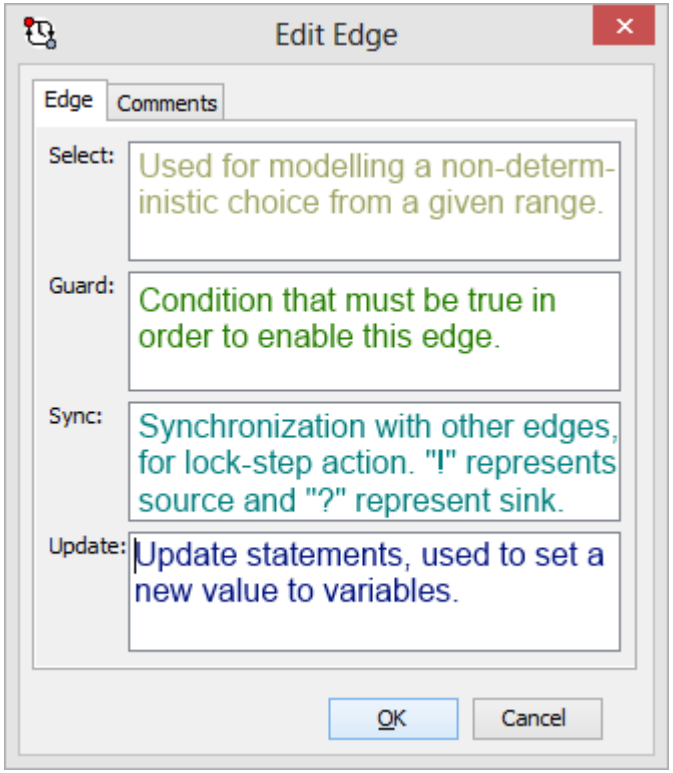
- Double click on the edge between “Safe” and “Appr” locations.
- Specify the Sync statement “appr[myId]!”.
- The “!” symbol specifies that this is a source for synchronization. Other sync edges using this channel must wait for this edge to be taken.
- Use the update statement to reset the clock variable.

Create the Train Component (9/10)



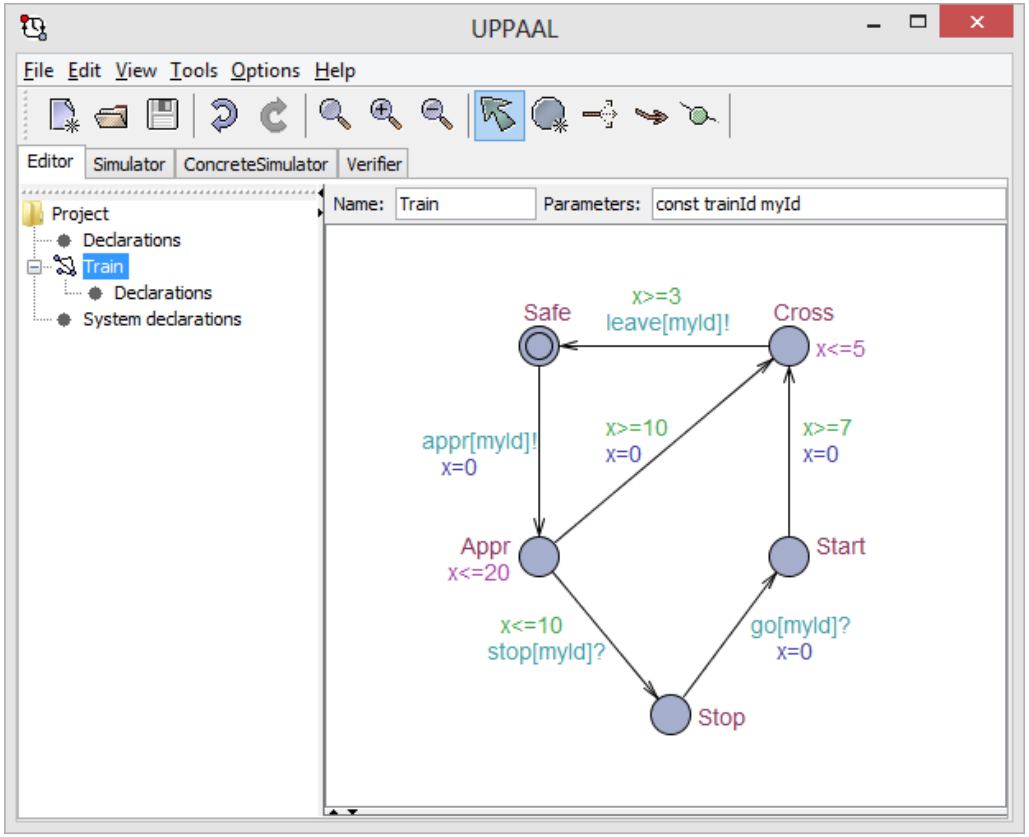
- Double click on the edge between “Appr” and “Stop” locations.
- Specify the Sync statement “stop[myId]?”.
- The “?” symbol specifies that this is a sink for synchronization and must wait for a source on this channel.
- Specify the guard for this edge, which is a condition that must be true before this edge can be enabled.

Editing Edges



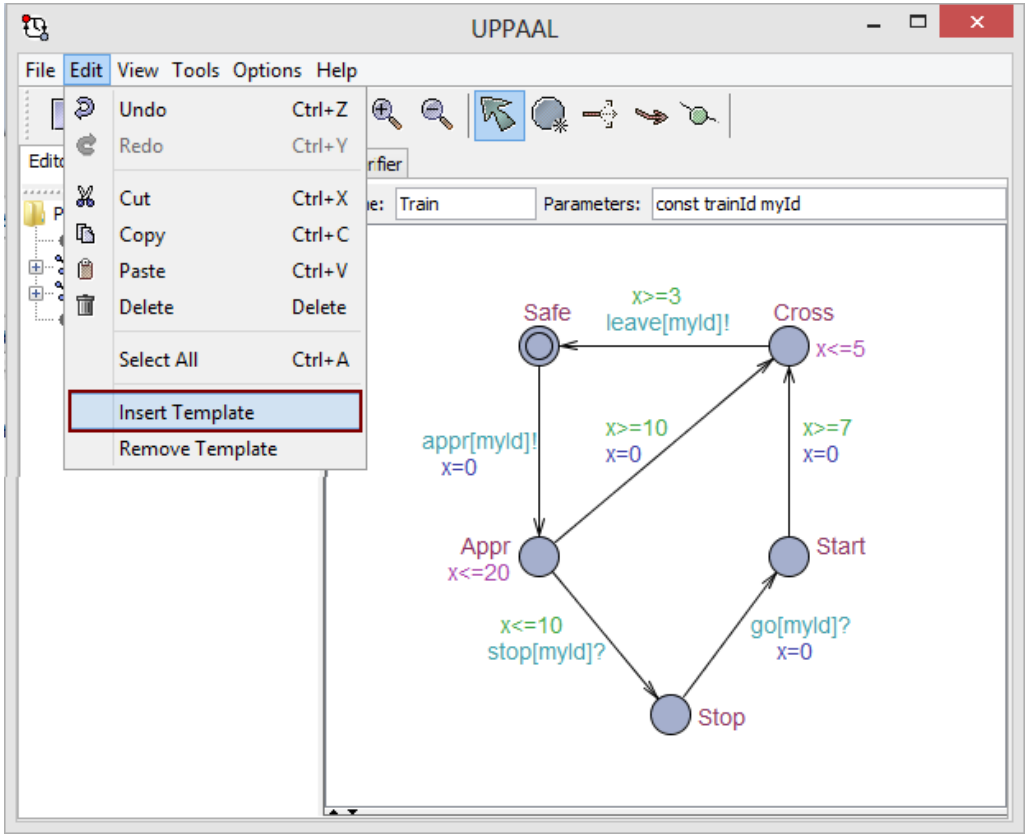
- Use “Selection Tool” to double click on an edge to bring up the “Edit Edge” dialog.
- The four components of an edge are described in the picture.
- Color coding of these components is used to visually differentiate between them.

Create the Train Component (10/10)



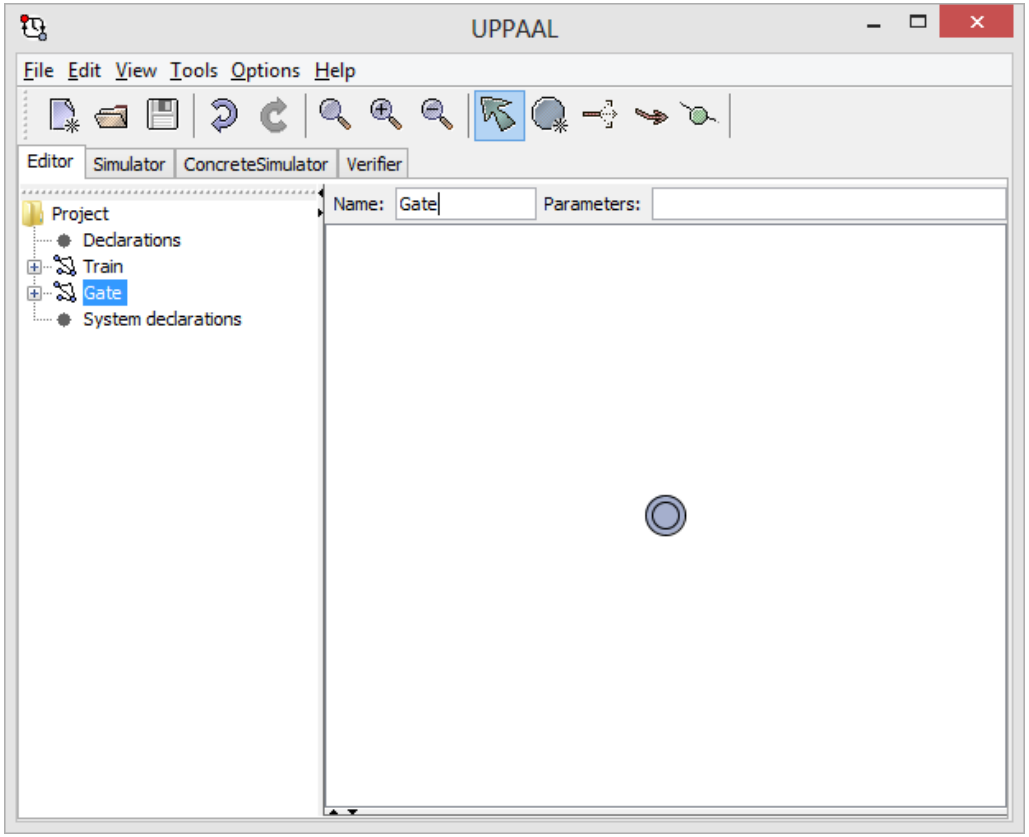
- Edit all edges of the Train component to match the given picture.
- Make use of the color coding to avoid mistakes.

Create the Gate Component (1/10)



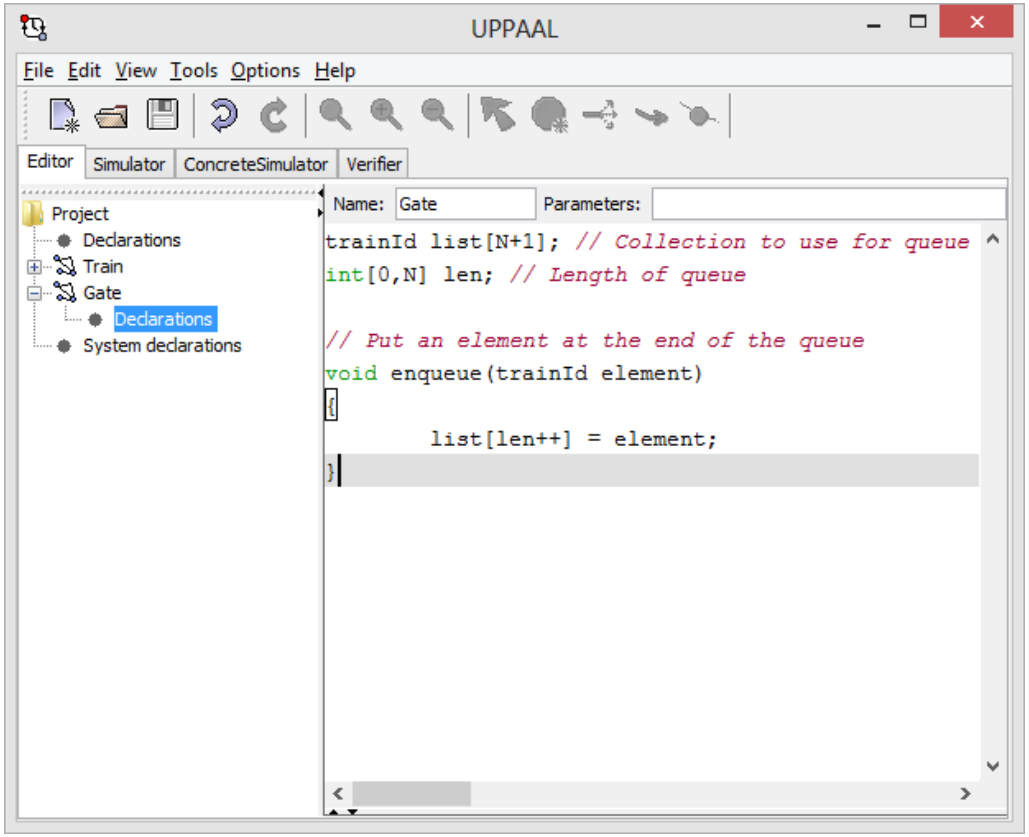
- Open “Edit” menu and choose “Insert Template”.
- This will create a new component in the project.

Create the Gate Component (2/10)



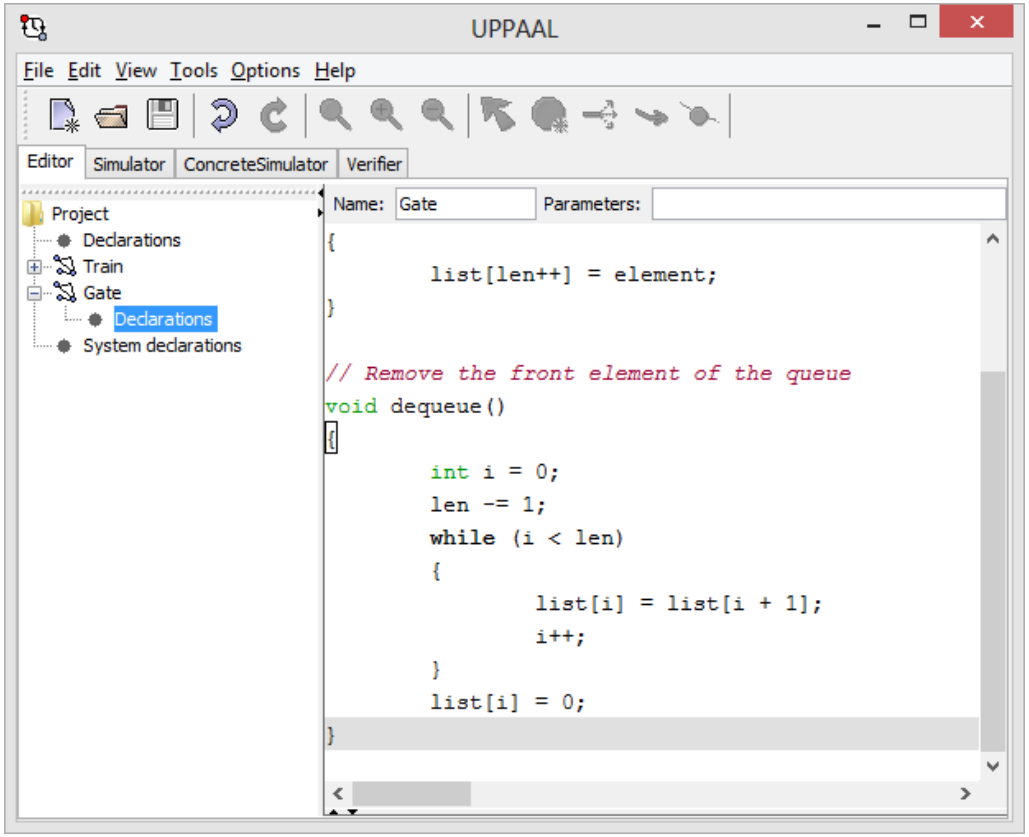
- This template comes with a pre-existing initial location.
- Rename the new component from “Template0” to “Gate”.

Create the Gate Component (3/10)



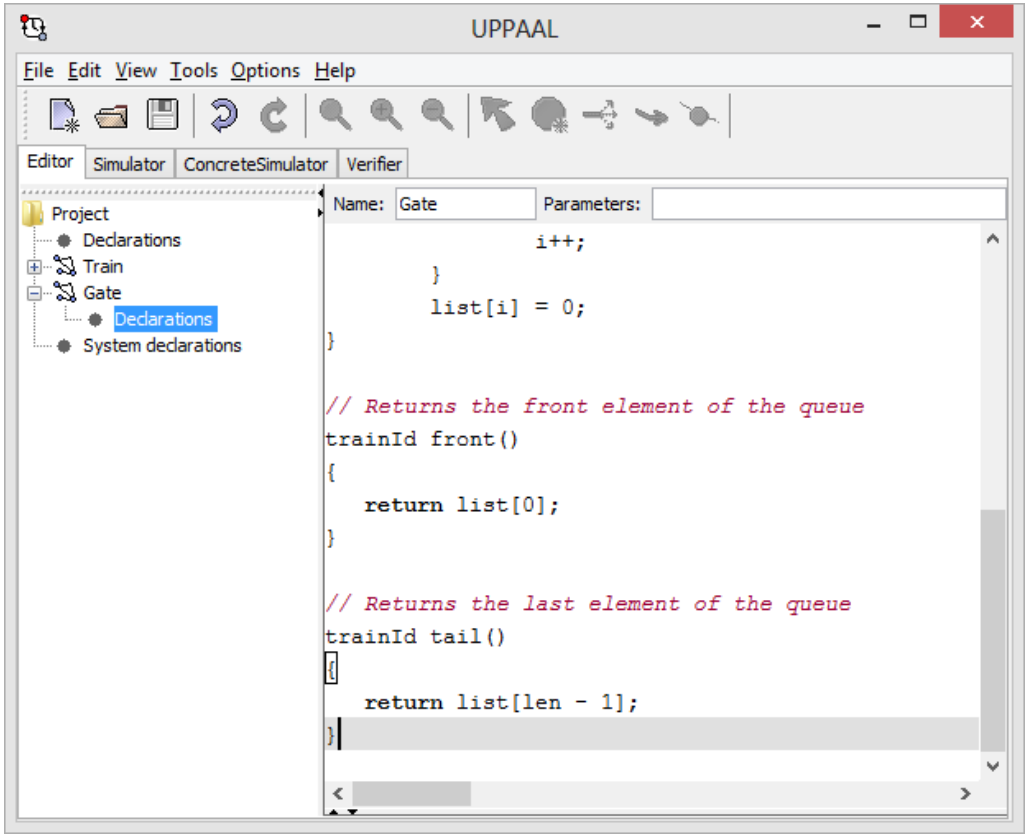
- Before we make the state machine of the Gate component, let us first implement a FIFO queue for servicing the train requests.
- Create an array to hold collection of elements of type trainId.
- Create an integer to maintain the current length of the queue.
- Create the “enqueue” method to add requests to the FIFO list and update the length.

Create the Gate Component (4/10)



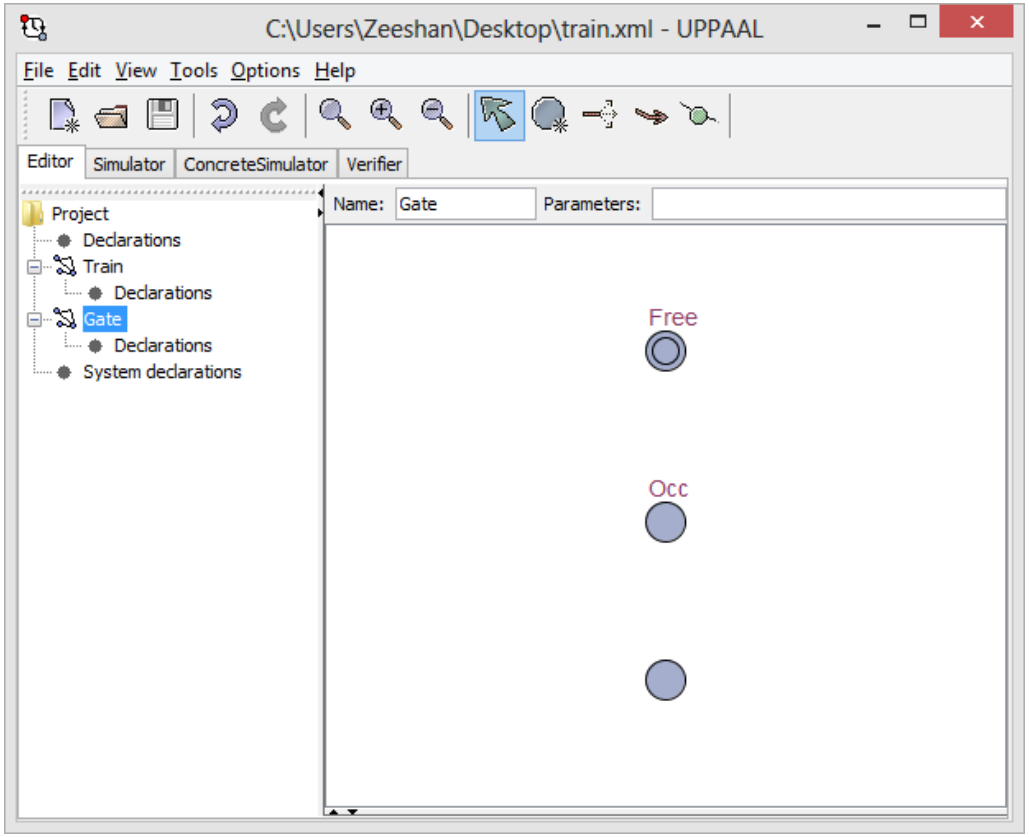
- Similarly, create the "dequeue" method to remove the first item from the list.
- The first item is always at the index zero.
- Shift all request an index below to simulate a FIFO queue.
- The emptied index is represented by a zero value.

Create the Gate Component (5/10)



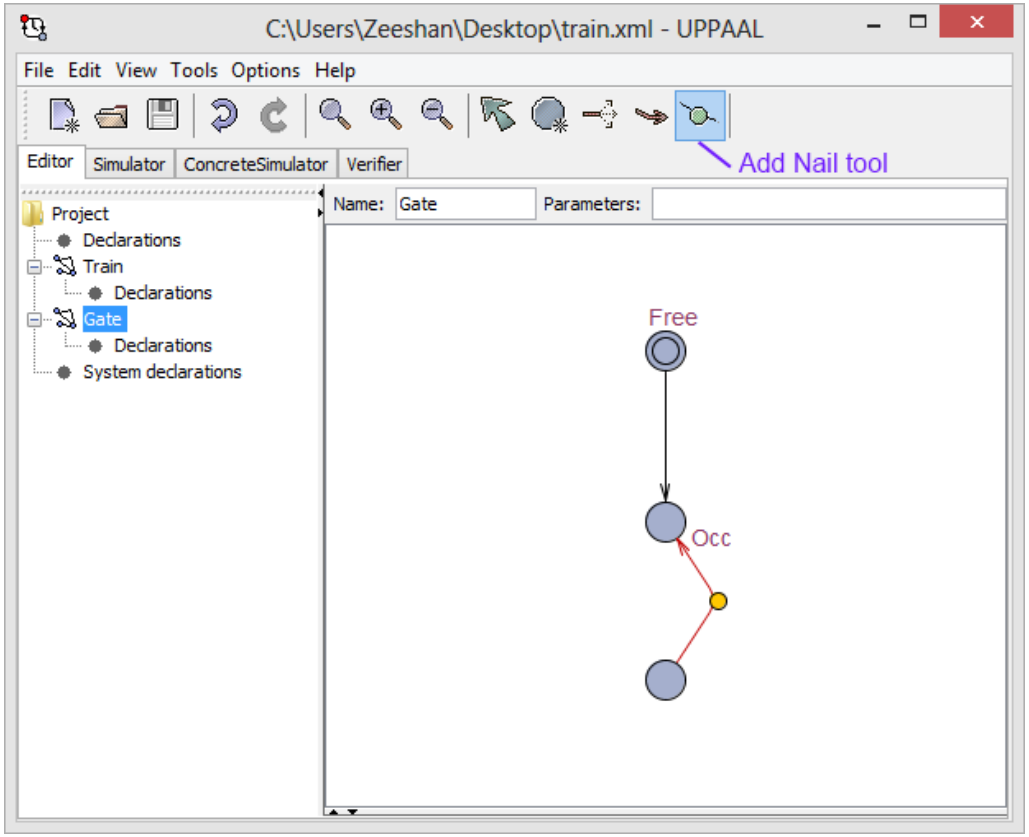
- Create the "front" and "tail" methods to peek at the first and last item in the list, without removing them.

Create the Gate Component (6/10)



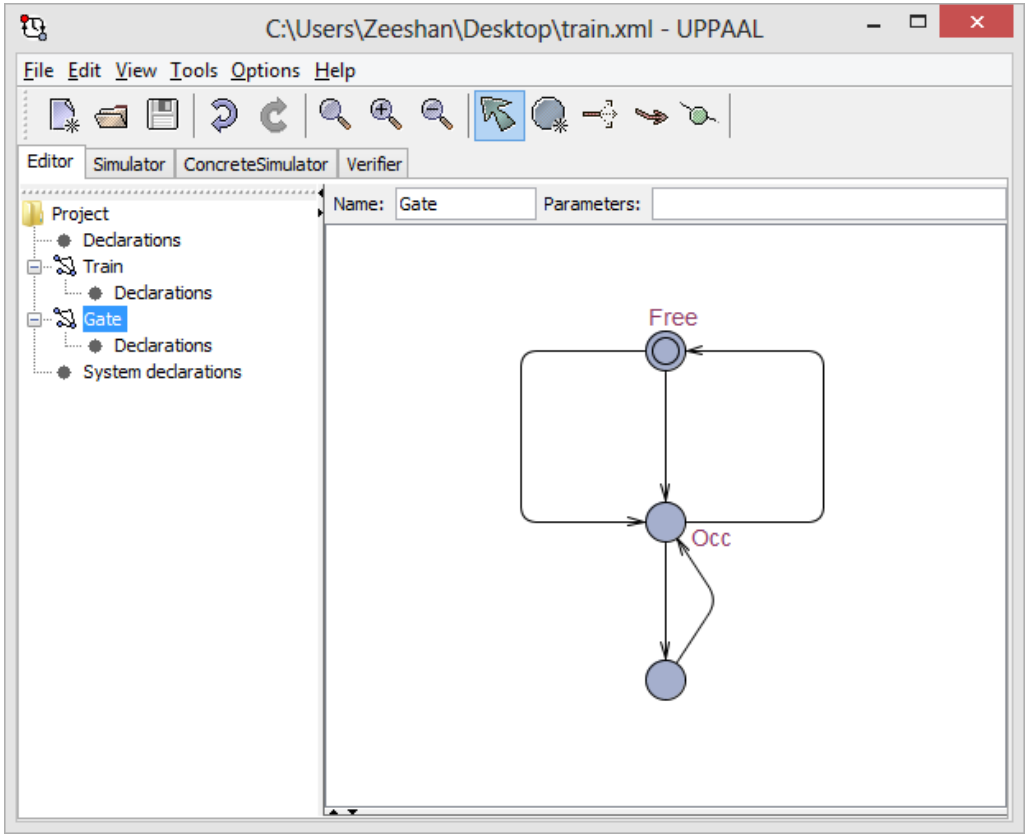
- Click on the “Gate” node in the project explorer.
- Use the “Add Location Tool” to create two more locations in addition to the pre-existing initial location.
- Use “Selection Tool” to arrange the locations as shown in the picture.
- Double click on location to set the respective names as shown in the picture.

Create the Gate Component (7/10)



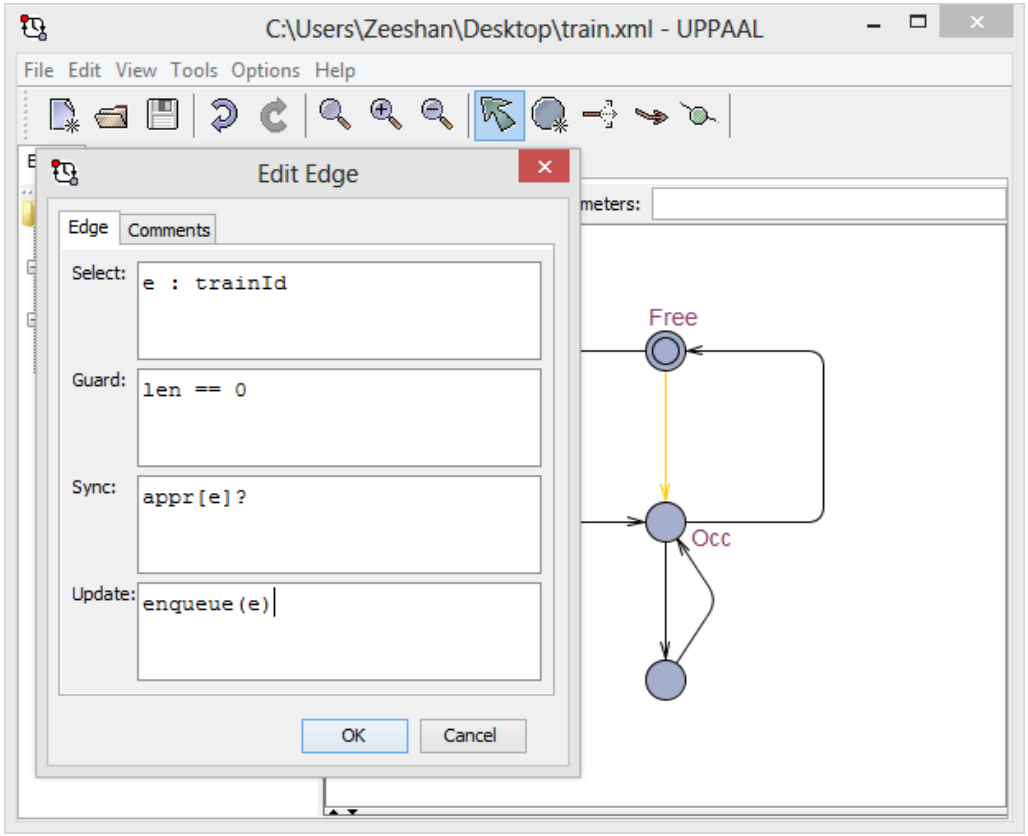
- Use the "Add Nail Tool" to create bent edges. This feature is useful where multiple edges exist between two given locations.
- Pick the "Add Nail Tool" and click on the edge.
- Pick the "Selection Tool" to drag the created nail, the edge will bend along with the nail.
- Multiple nails may be added to an edge.

Create the Gate Component (8/10)



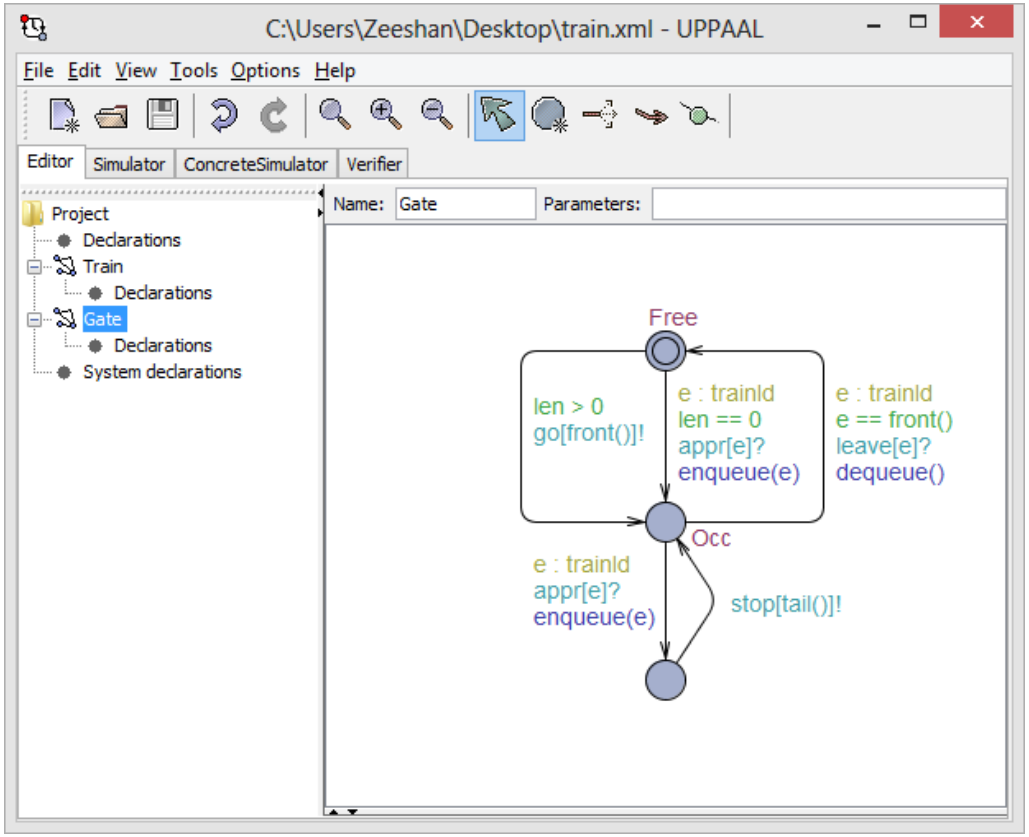
- Use the “Add Edge Tool” and “Add Nail Tool” to create edges as shown in the picture.
- Ensure that the direction of edges is correct.

Create the Gate Component (9/10)



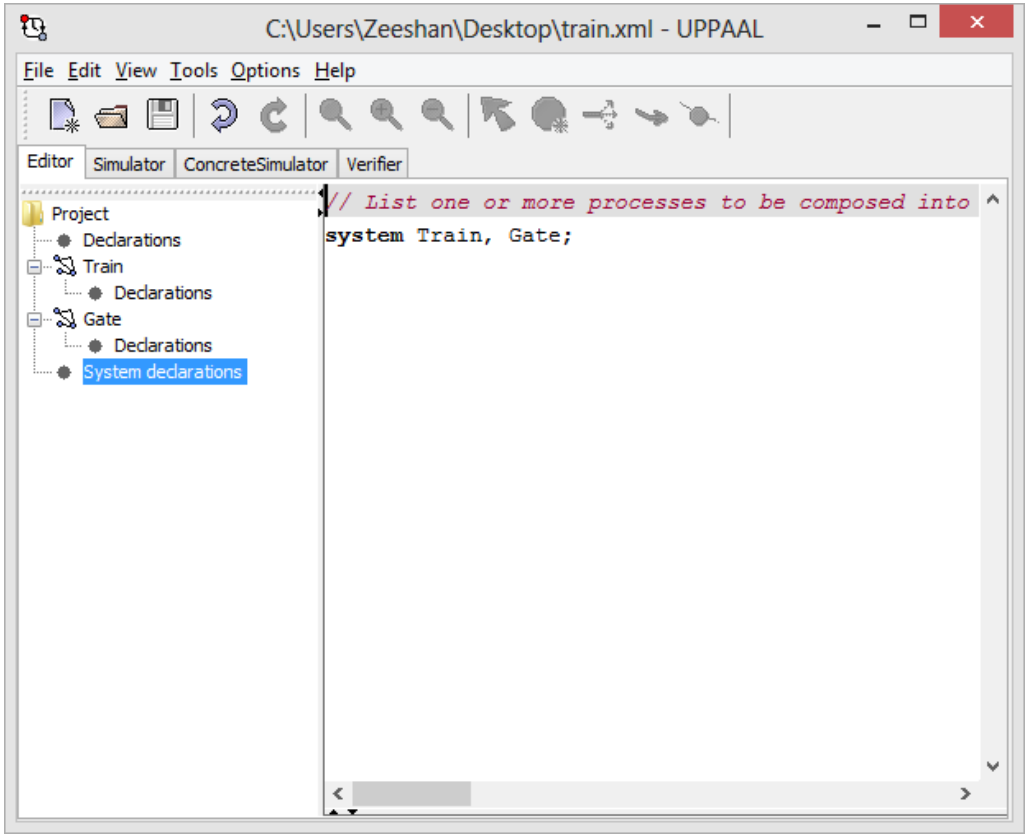
- Edit the edge and provide the shown four components.
- "Select" enables non-deterministic choices. In this edge, the value of e is non-deterministically chosen from all possible values of type "trainId" i.e., [0,5].
- This edge waits for an approaching train (any of 6 trains) and then enqueues this as a request.
- This edge is only enables when queue is previously empty.

Create the Gate Component (10/10)



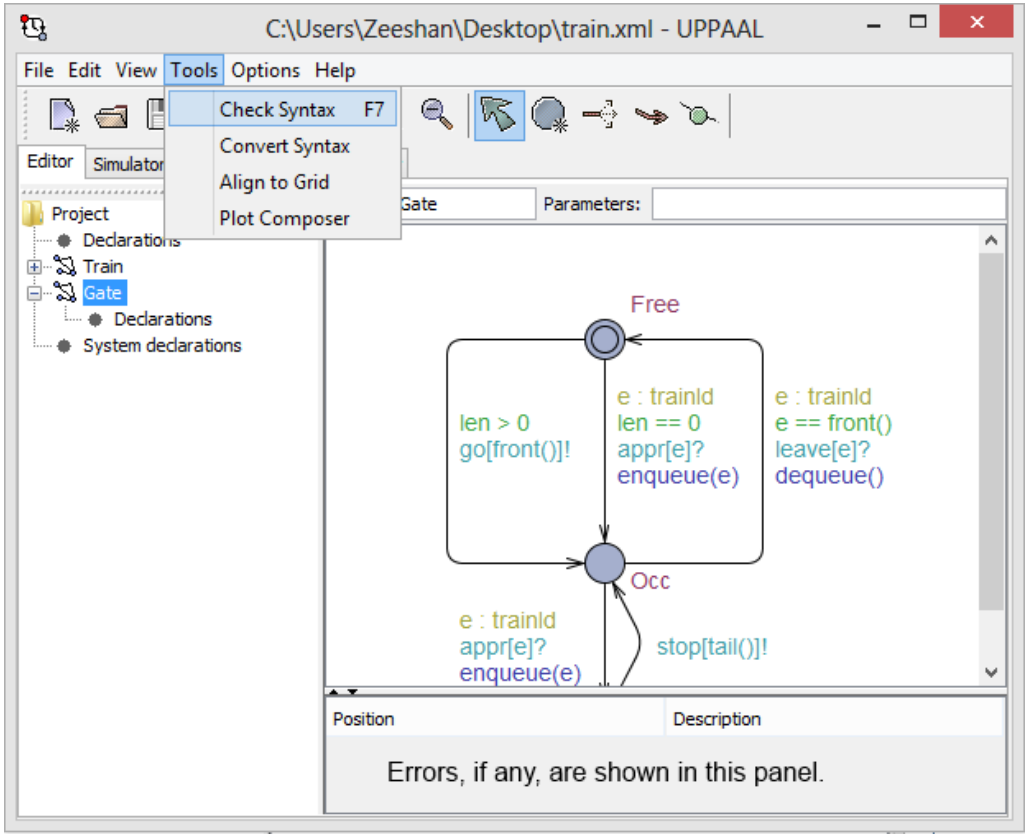
- Similarly, add details to all edges as shown in the picture.
- Make use of the color coding to validate your edges.
- The T.A. will ask you explain some of these edges. Make sure you understand the purpose and behavior of these edges.

System Declarations



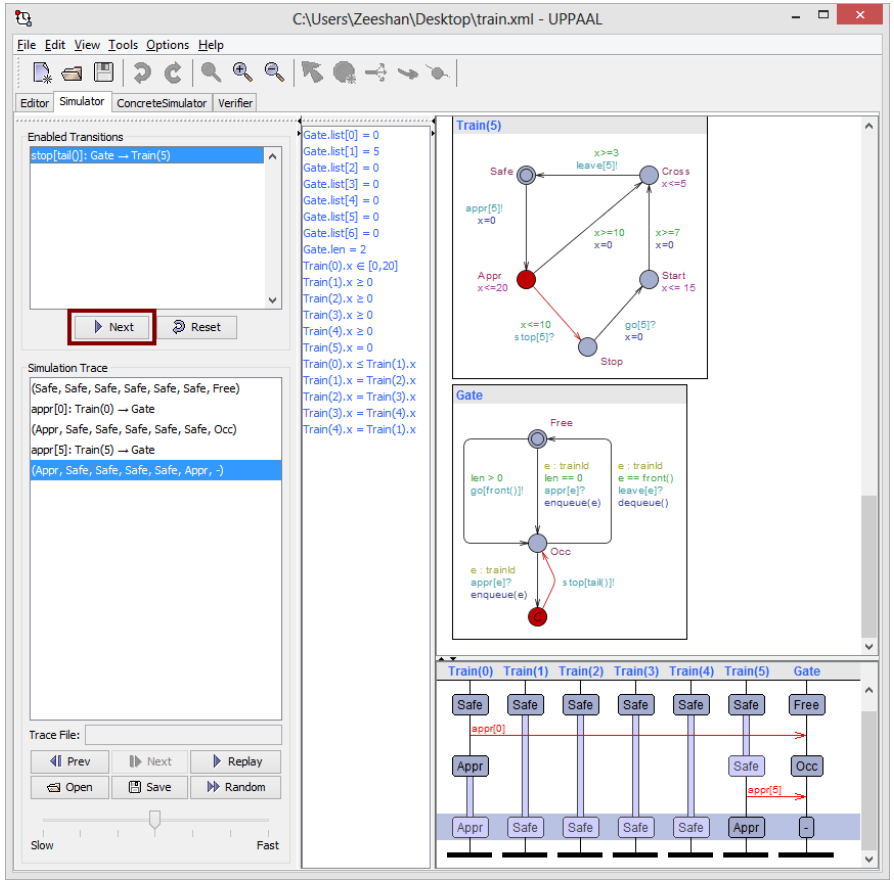
- Compose the system with the two components “Train” and “Gate”.
- This tries to initialize the respective components with parameters (if any).
- Since “Train” uses a “trainId” parameter and it is not specified with a value, all possible values of “trainId” ([0,5]) are used to create (6) copies of “Train” component.

Save and Check Syntax



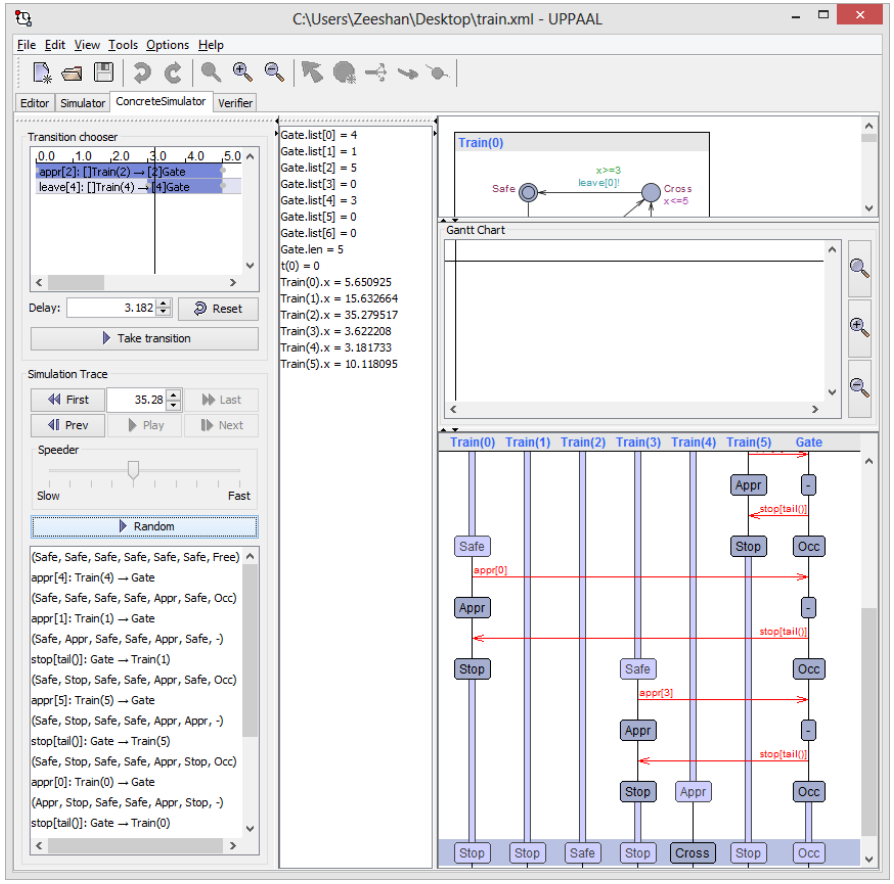
- Save the file in a writeable location.
- Use the "Tools" menu and select "Check Syntax" or press F7.
- You should not get any errors.

Simulation



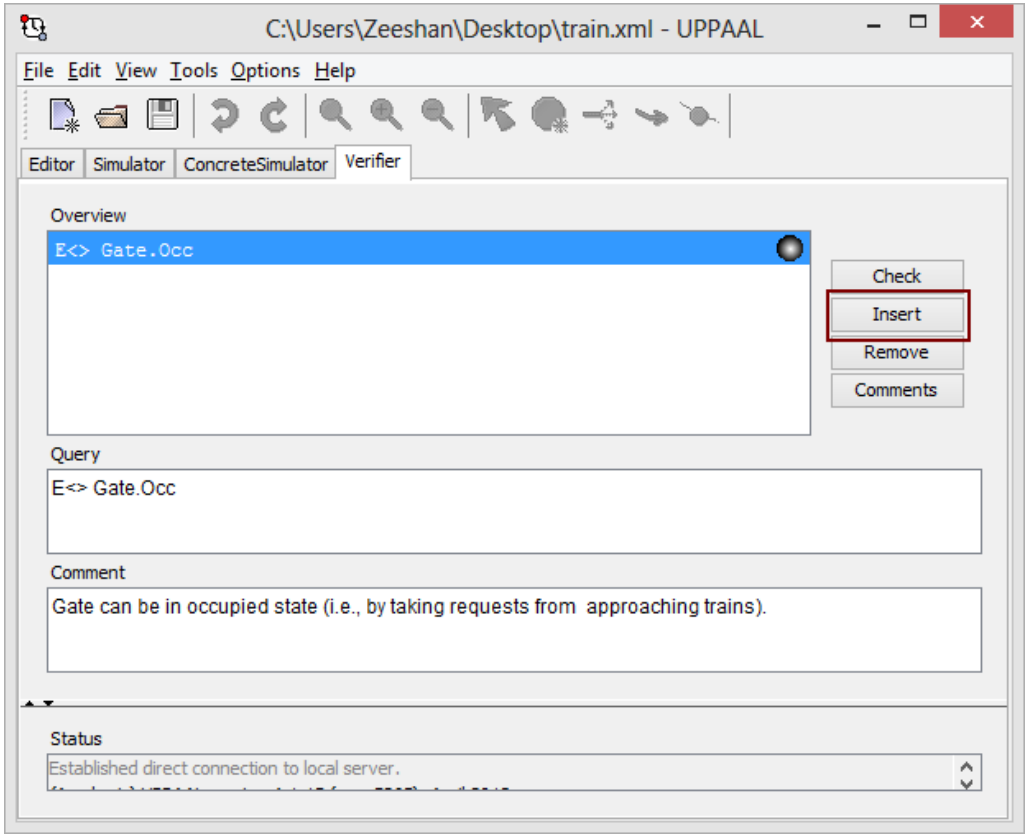
- Switch to “Simulator” tab.
- The system will initialize the components and begin with the initial location.
- Click the “Next” button a few times to view how the system progresses.
- The current state of the system is shown in the white column in the middle. Click on list items in the “Simulation Trace” to view how the simulation progressed.
- Find out the purpose of each of the shown pane in this view.

Concrete Simulation



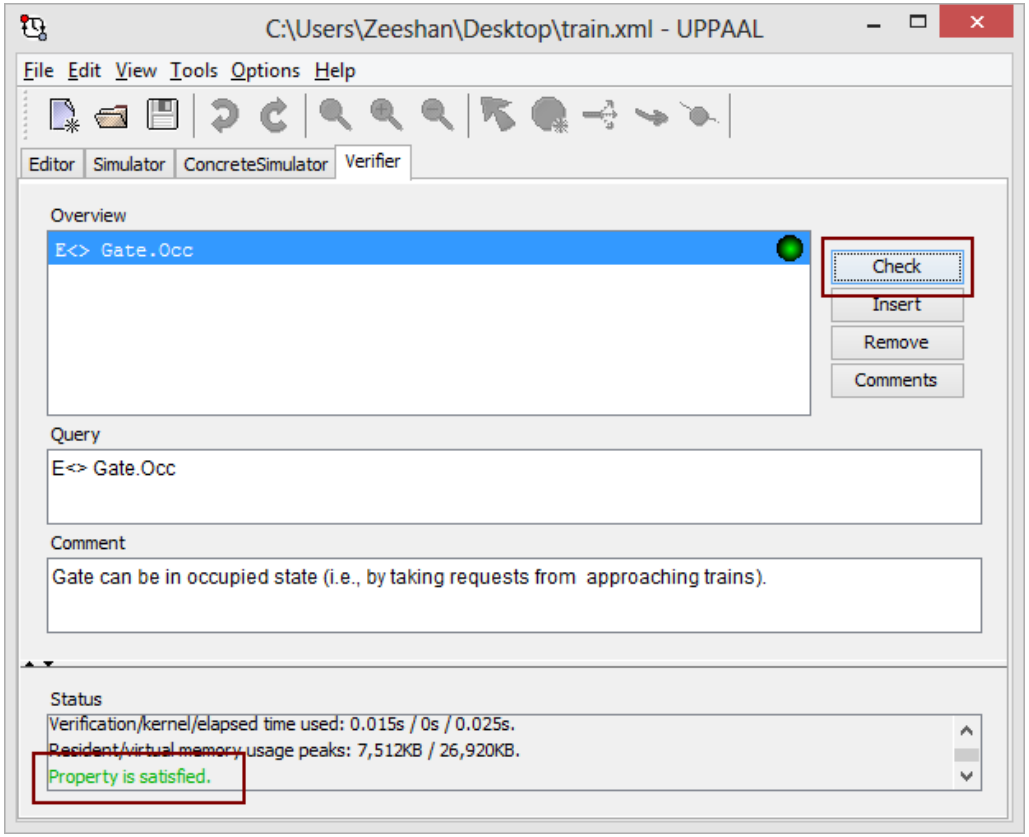
- Concrete simulation uses concrete values rather than symbolic values.
- Click “Random” button to start concrete simulation. Pause the simulation after a few iterations by clicking the same button again.
- Compare the values of clock variables (e.g., “Train(0).x”) between the two types of simulations.
- Experiment with the Numeric UpDowns in the left panel.

Verification (1/2)



- Click "Insert" to create a new verification property.
- Type "`E<> Gate.Occ`" in the query field.
- Write a meaningful comment in natural language.

Verification (2/2)



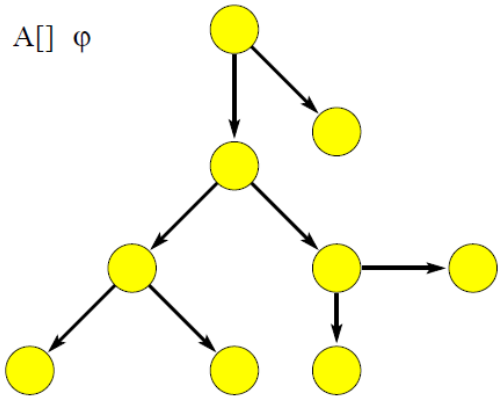
- Click the property in the "Overview" pane, click "Check" to verify this property.
- The verification result and statistics are visible in "Status" pane, as well as shown as a green/red icon in the "Overview" pane.

CTL in UPPAAL

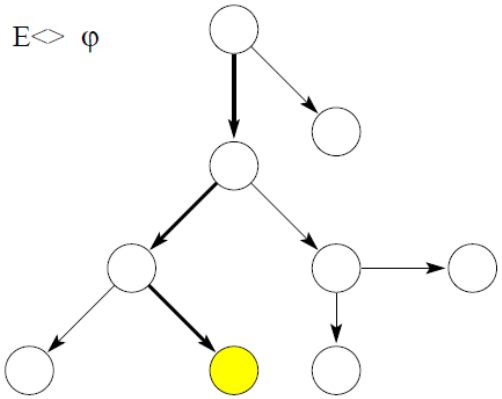
- E : exists a path
- A : all paths
- <> : eventually / finally
- [] : always / globally
- --> : implies
- not : negation
- forall (i : [l,u]) : for all values of i in the given range.

Examples of A/E and []/<>

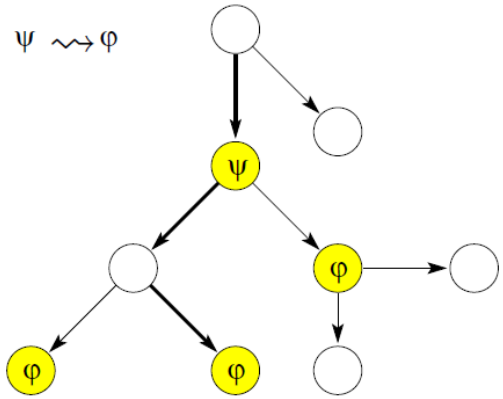
A[] φ



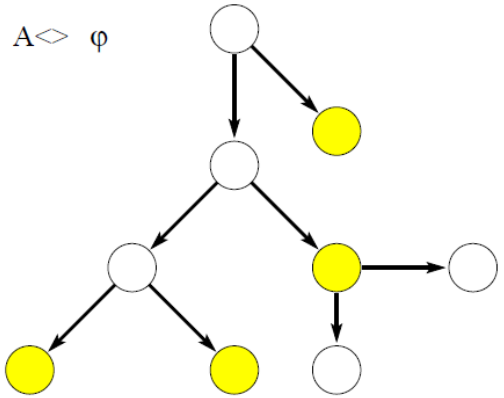
E<> φ



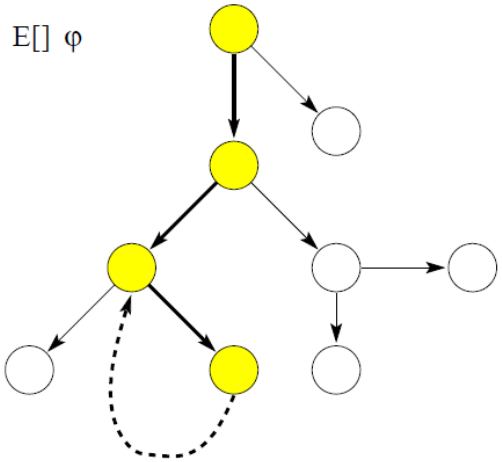
ψ ⇝ φ



A<> φ



E[] φ



Verify the following properties

- Train 2 can cross.
- A situation is possible that Train 0 is crossing, while Train 1 is waiting.
- When Train 0 is passing, all other trains must wait.
- The wait queue never overflows.
- The wait queue never underflows.
- Whenever a train approaches, it eventually crosses.
- System is deadlock free.
- There is never more than one train crossing at a time.
- Create a property from your understanding of the system and verify it.

Find and fix errors

- There are **two** errors in the system. Find and fix these.
  - Hint : use verification to find a property that fails and check why.

Further Help on UPPAAL

- UPPAAL Tutorial
  - http://doc.utwente.nl/51010/1/Tutorial-UPPAAL-Behrmann.pdf