Introductory Tutorial

Eugene Syriani

In this tutorial, you will learn to…

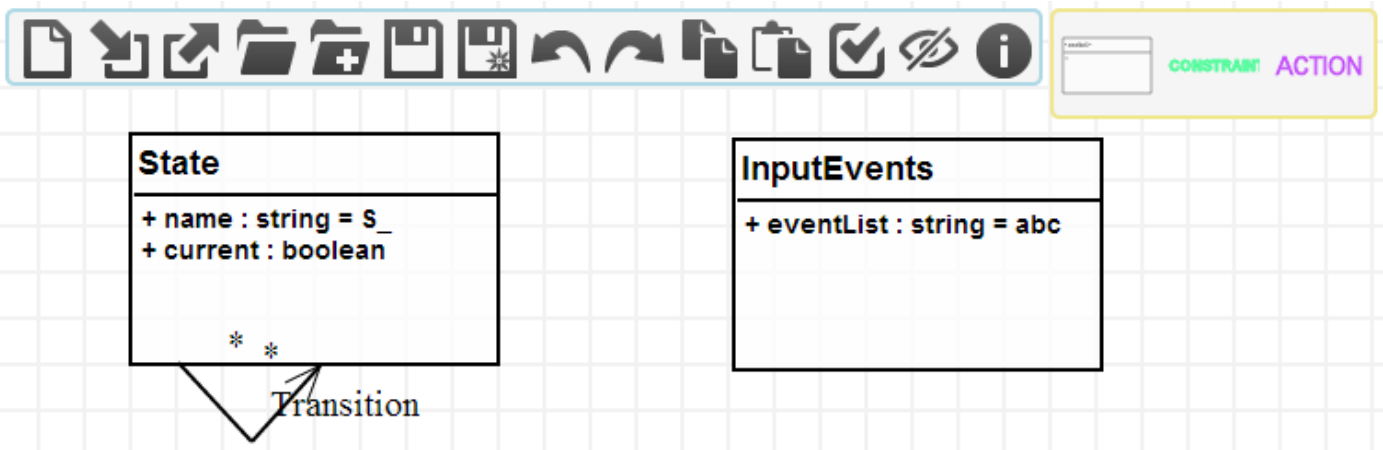
- Create a domains-specific language for modeling **finite state automata (FSA)**
- Synthesize a **modeling environment** for designing FSA models
- Design a **simulator** for FSA

For that, we will:

- Define the abstract syntax
- Define the concrete syntax
- Define a model transformation for simulation

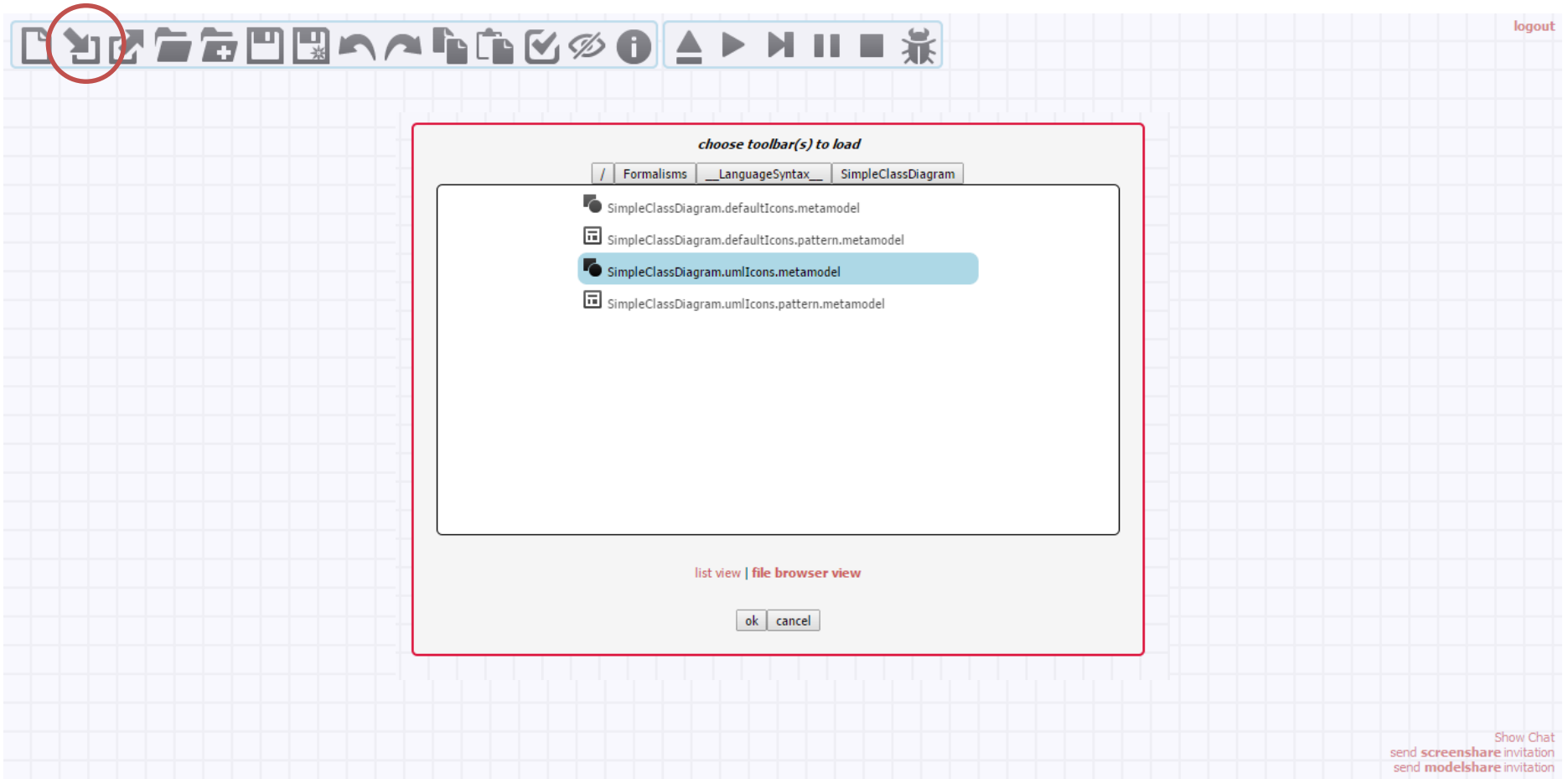
Abstract syntax

- The first step is to build the metamodel
- We can use class diagrams to do that



Build the metamodel

Load the `SimpleClassDiagram` formalism toolbar



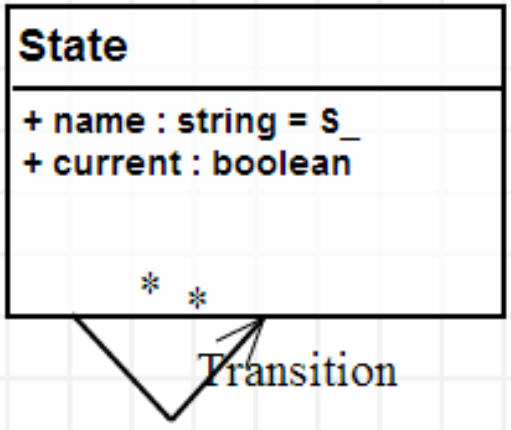
Build the metamodel

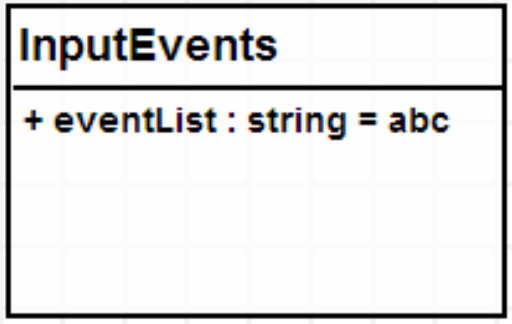
Canvas actions

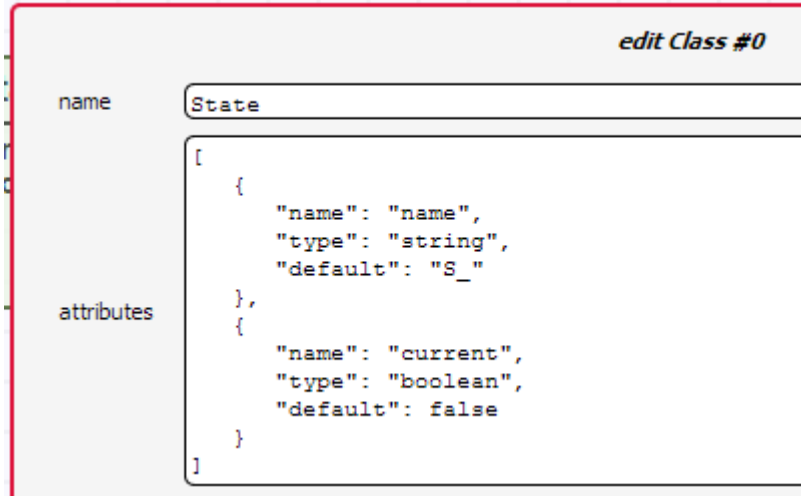
- Click on the `Class` icon in the toolbar
- Right click on the canvas to create a class
- To edit its properties, middle click on the class with no selection, or select the class and press `<INSERT>`
- Click `OK` to apply the changes
- To create an association, right click on the source class and drag your mouse to the target class (which can be the same class), then release your mouse
- Select the association and press `<SHIFT>` to move its control points
- Select an element and press `<CTRL>` to scale or rotate the element
  - Move the cursor to the icon corresponding to the action you want to perform
  - Scroll down or up on that icon. When you are satisfied, click on the checkbox

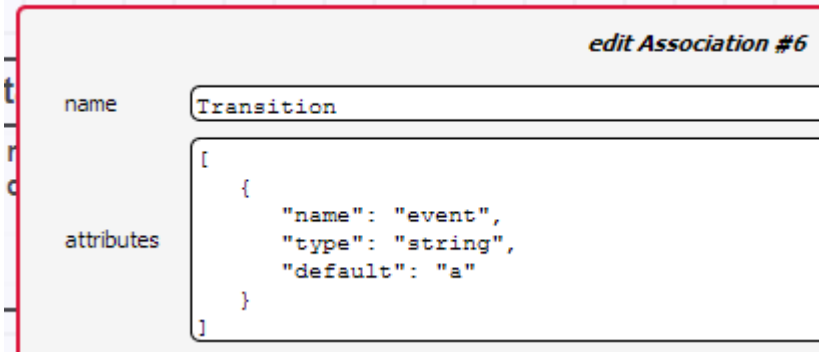
Build the metamodel

Draw the metamodel



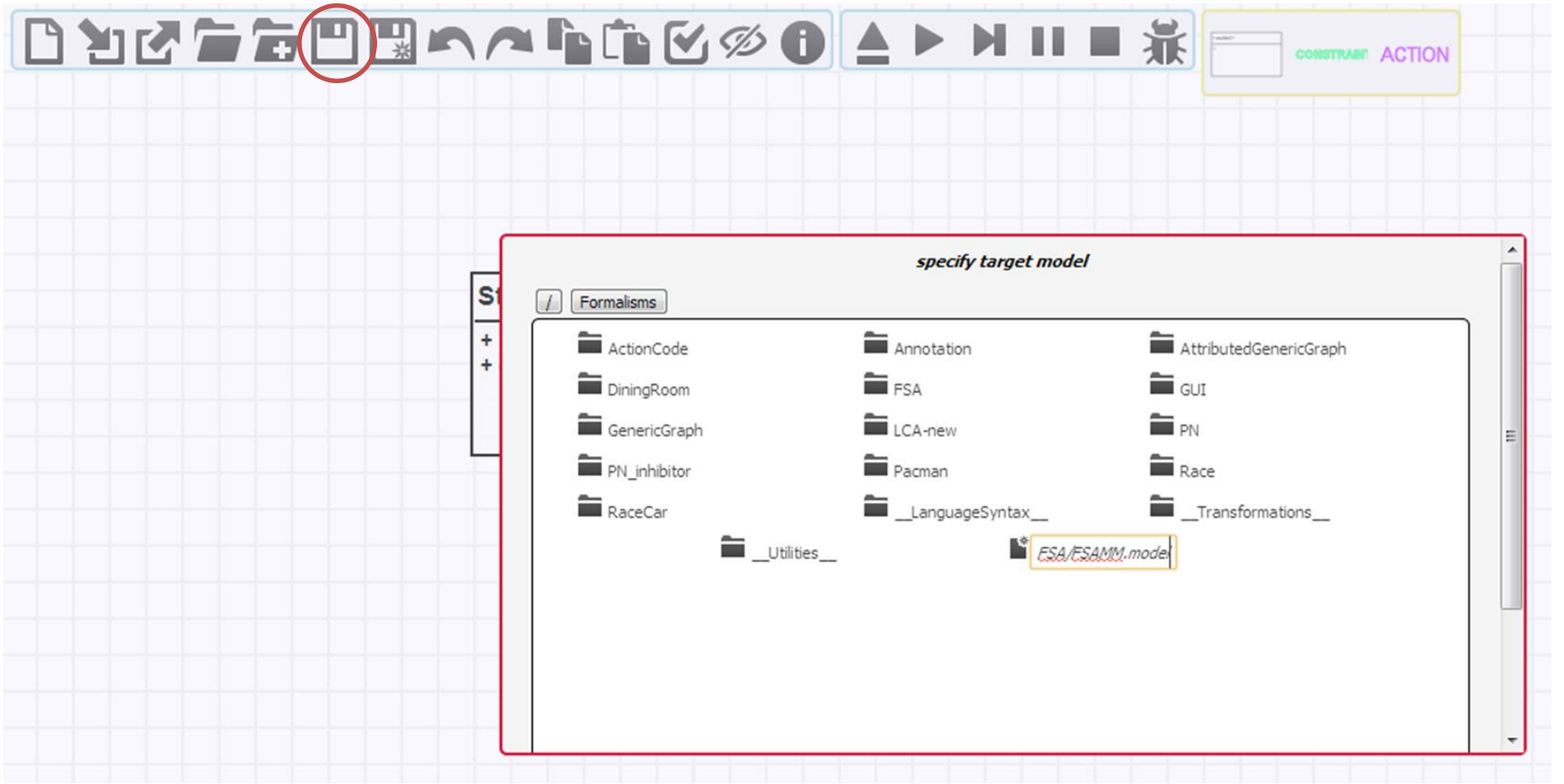






Build the metamodel

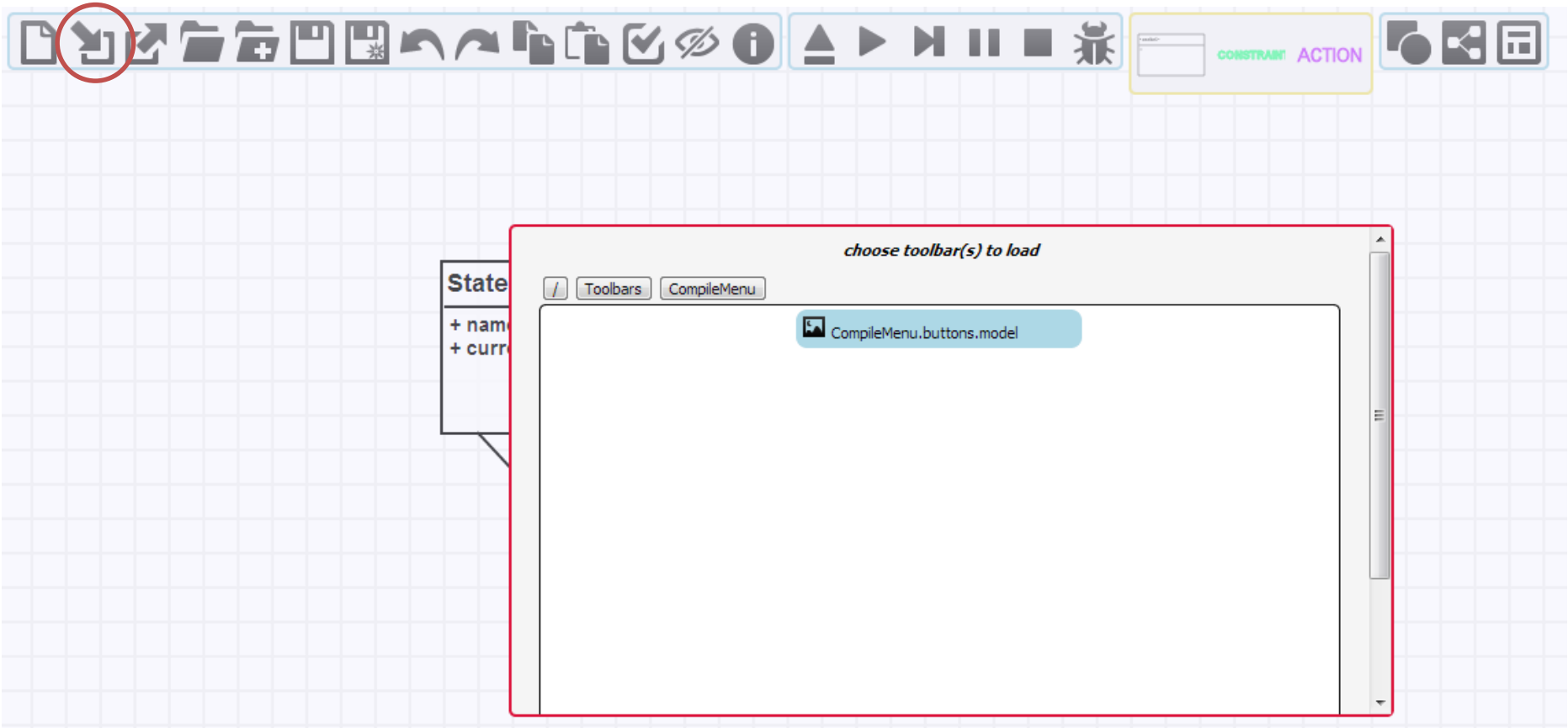
Save as your model under a new folder `FSA`



The file name must be `NAME+"MM.model"`

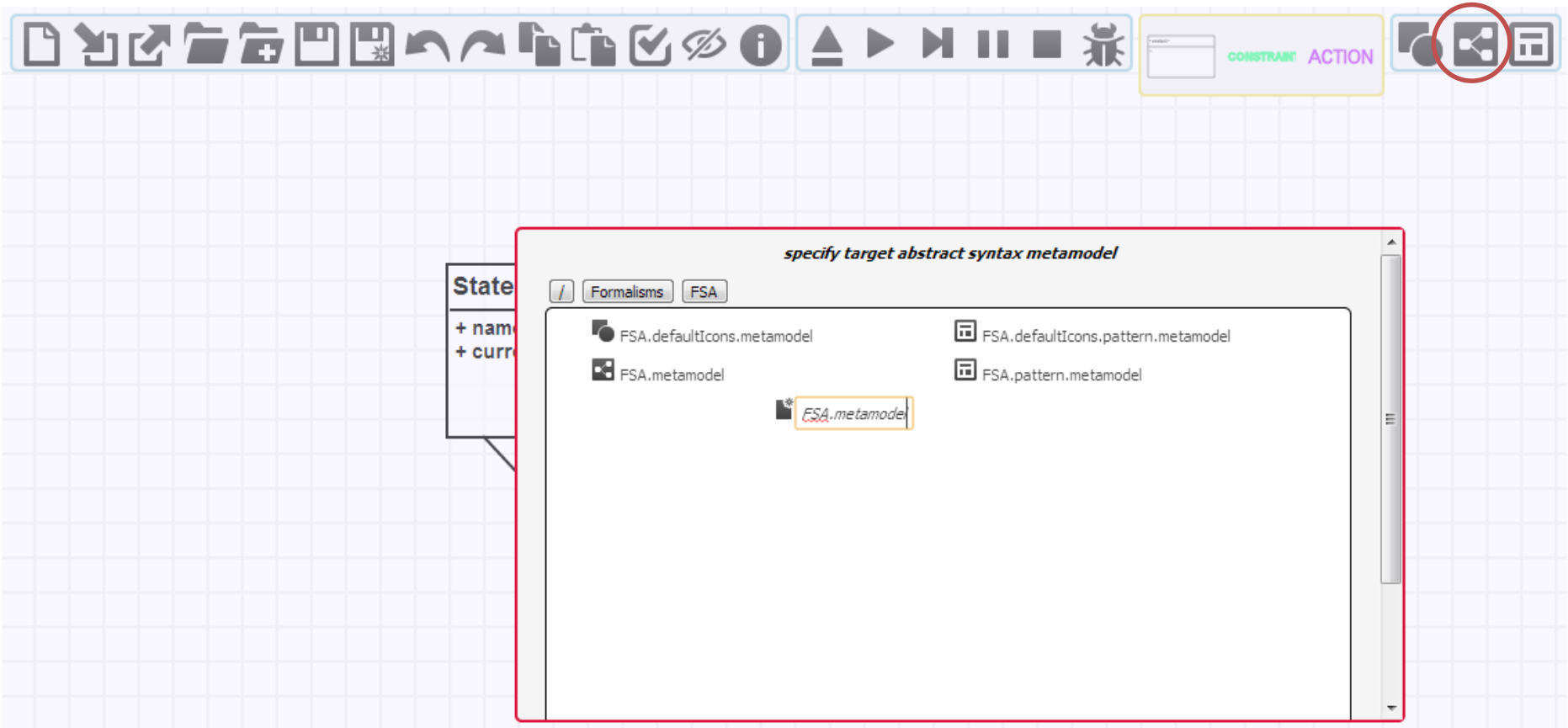
Build the metamodel

Load the `CompileMenu` toolbar



Build the metamodel

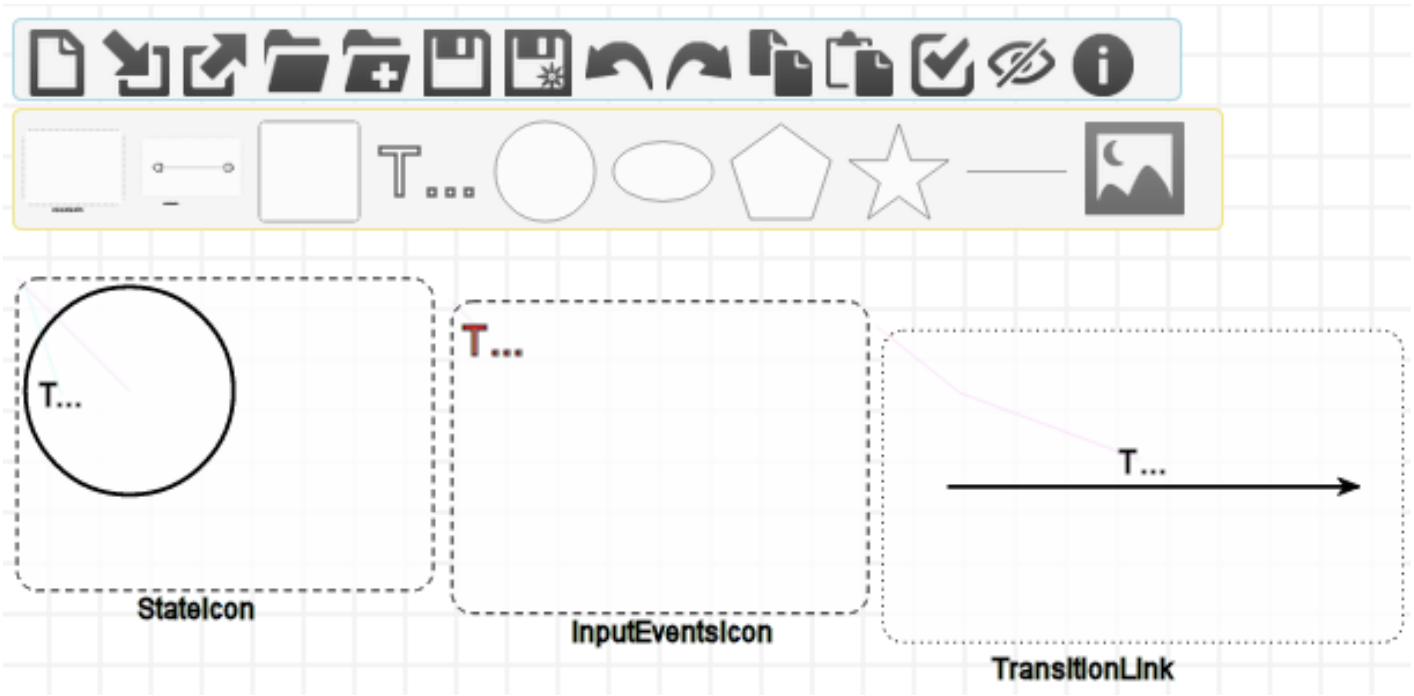
Generate the metamodel from this class diagram



The file name must be `NAME+".metamodel"`

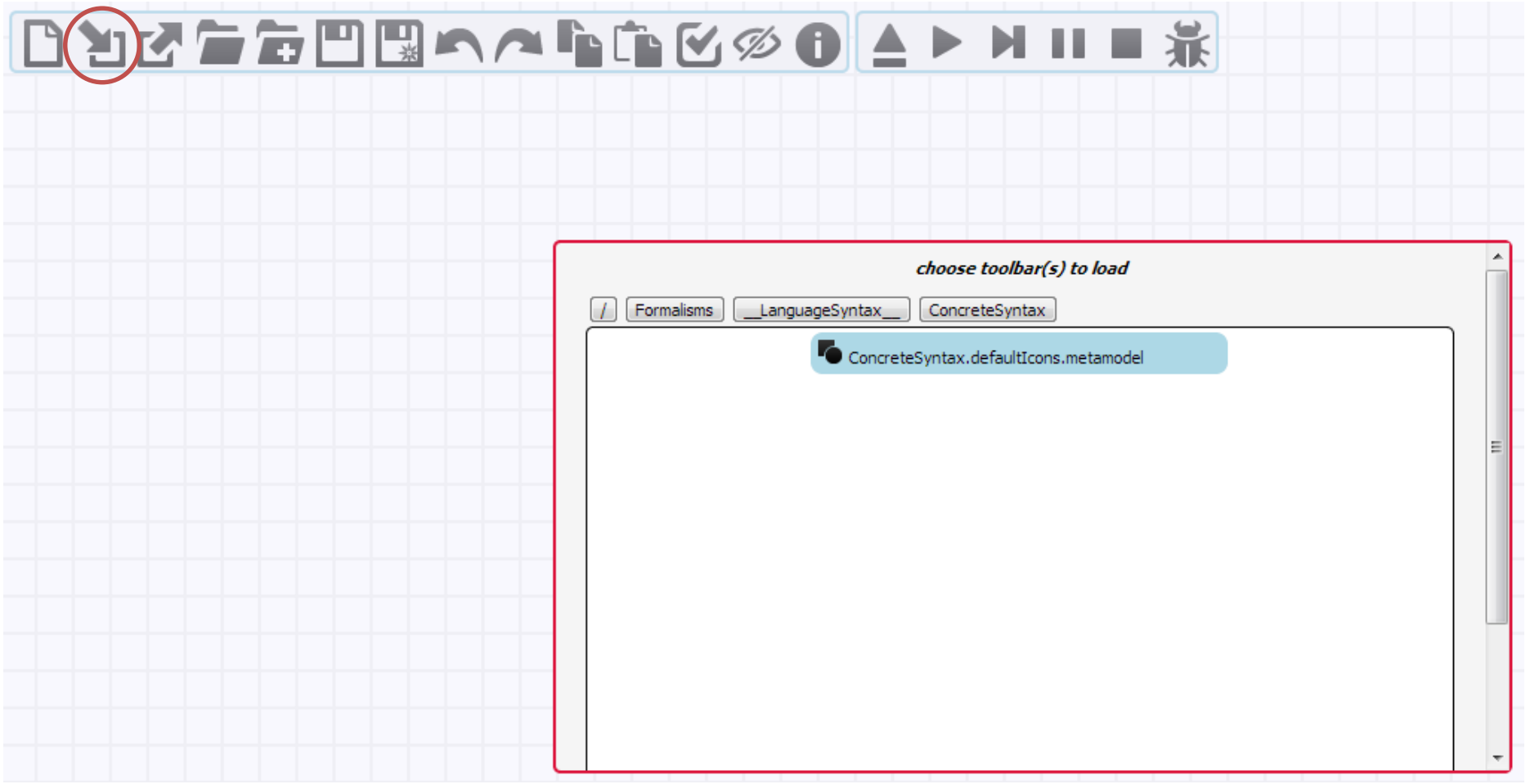
Concrete syntax

- Then we assign one possible concrete syntax model to the metamodel
- We can do that by drawing some shapes



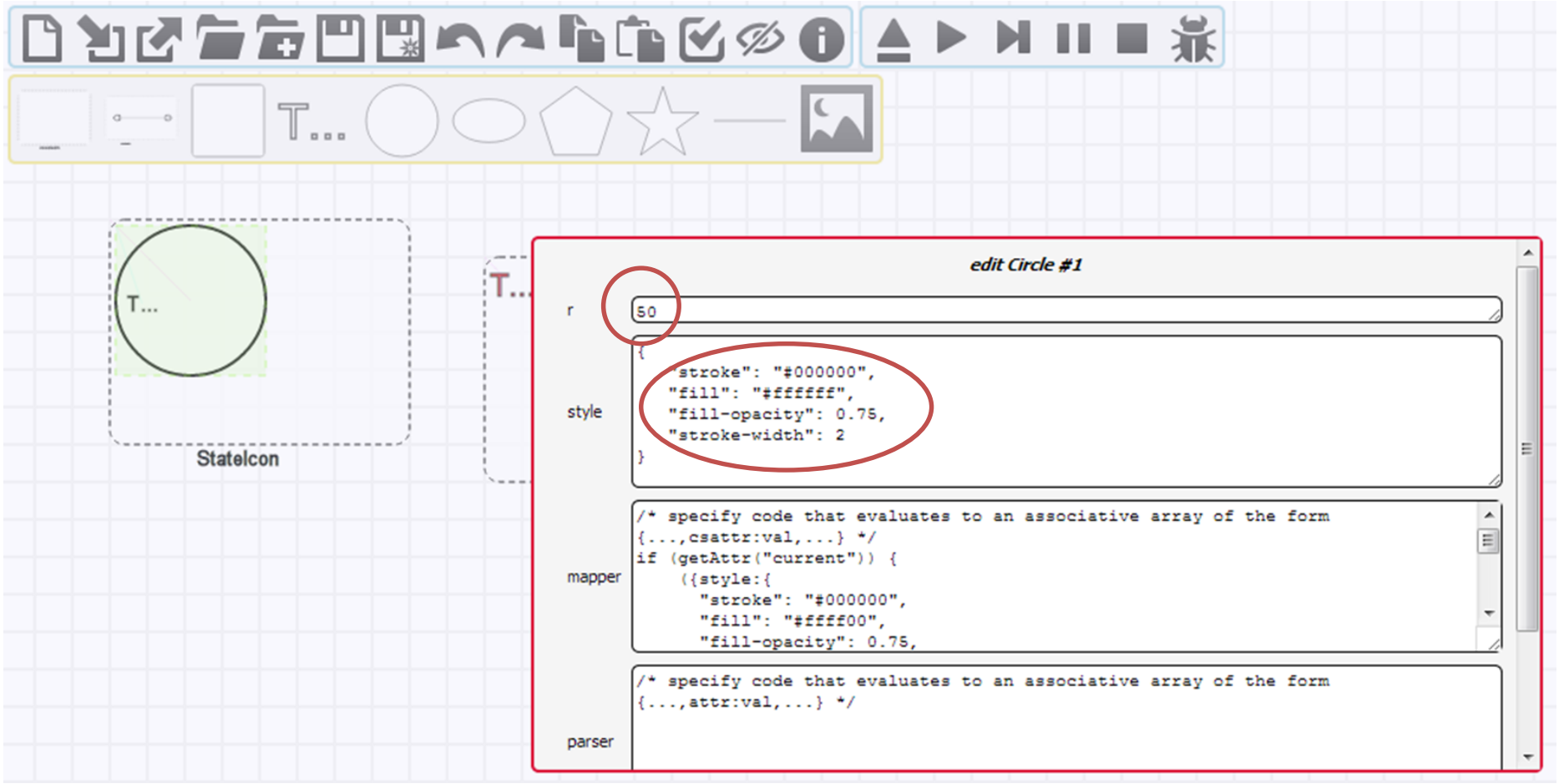
Concrete syntax

Load the ConcreteSyntax formalism toolbar



Build a concrete syntax

Draw the concrete syntax model



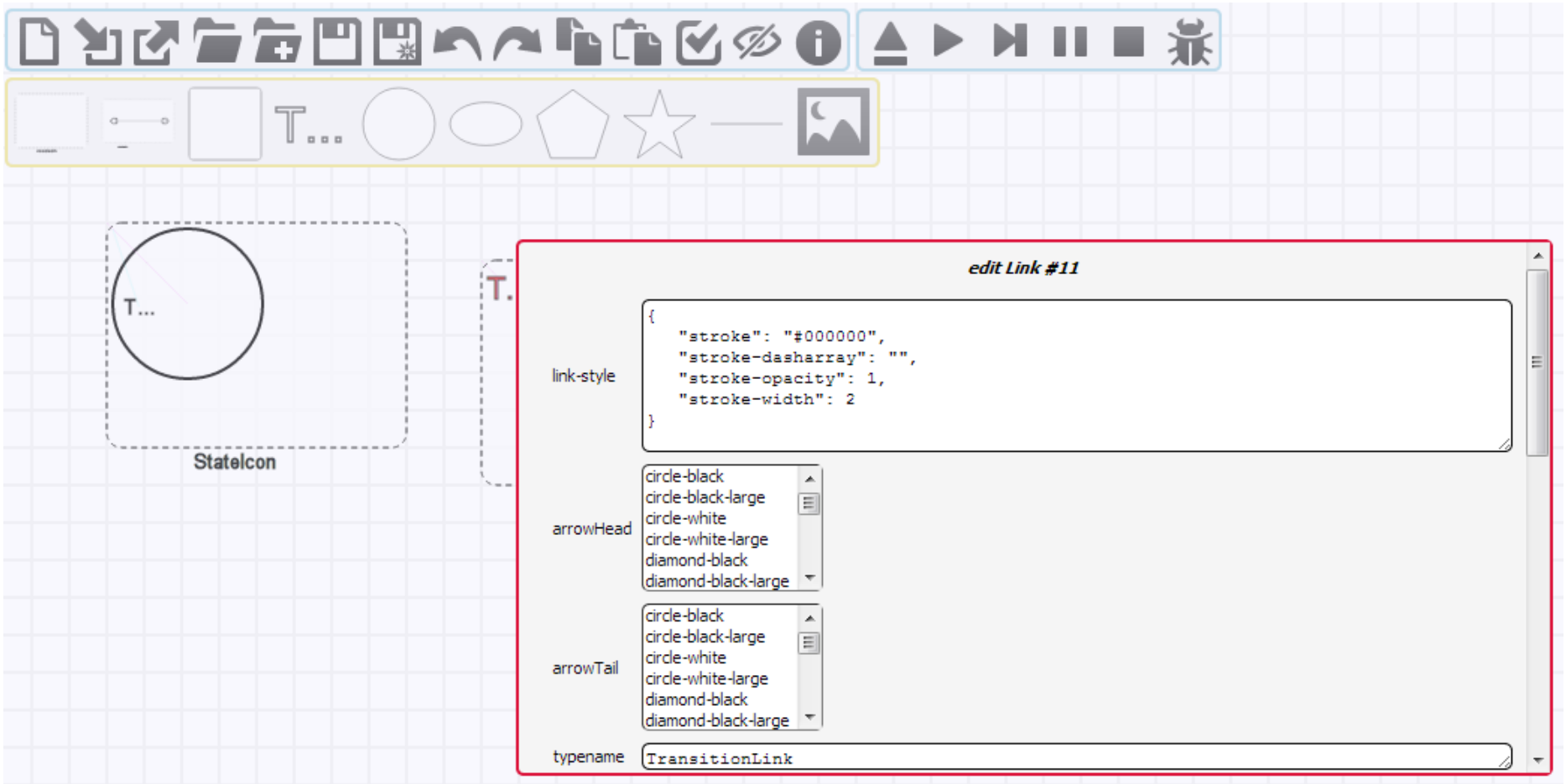
Build a concrete syntax

Drawing steps

- Create an `Icon` instance on the canvas for classes of your metamodel
- Its `typename` attribute must be in the form: `CLASSNAME+"Icon"`
- Create the shape you want outside the icon
- Modify resize it using its SVG attributes
- You may have multiple shapes for the same class icon
- Select the (group of) shape(s) and drag-n-drop it inside the icon close to the top left corner
- For associations, create a `Link` instance on the canvas
- To display some text, create a `Text` instance

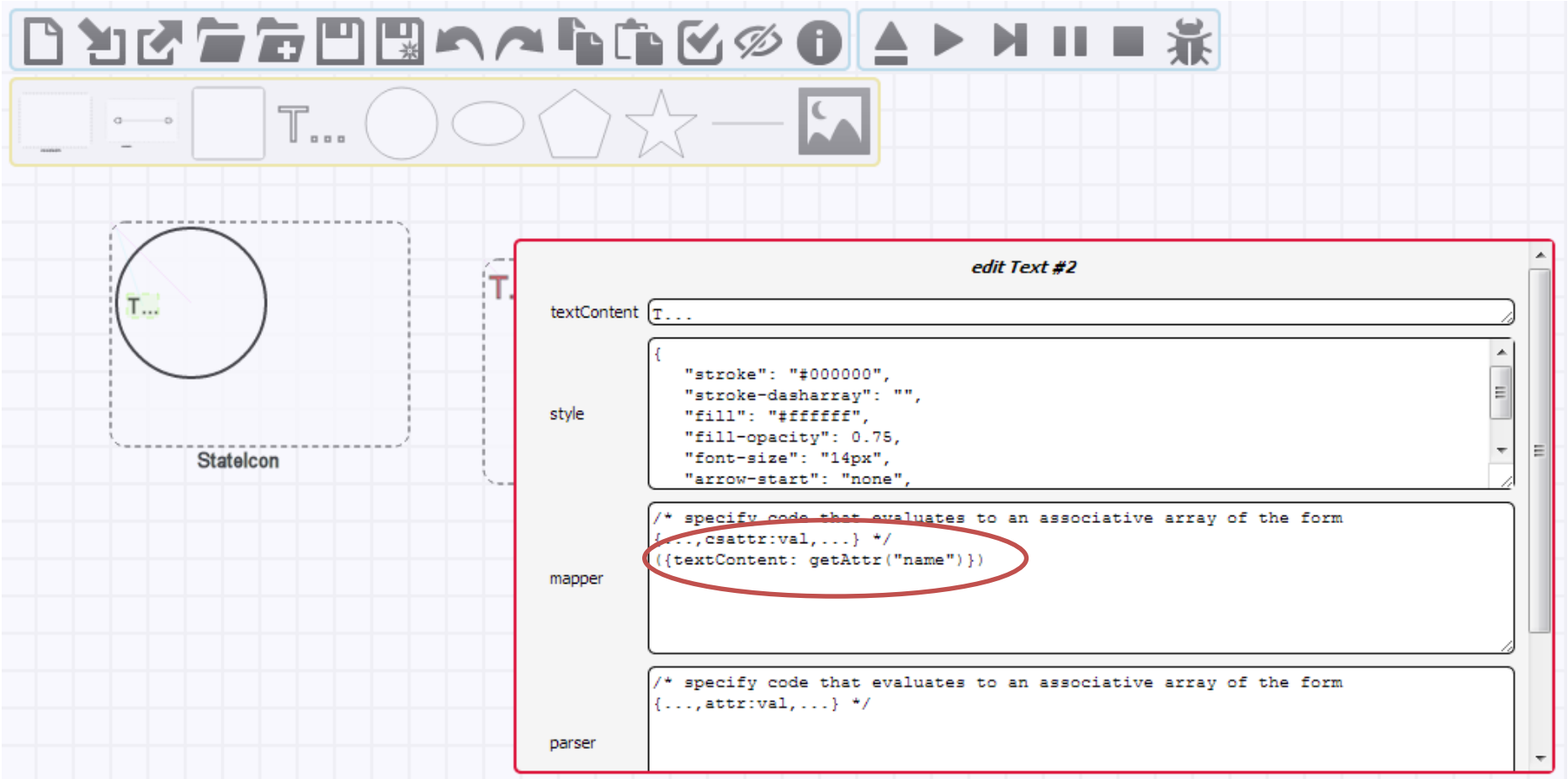
Build a concrete syntax

Draw the concrete syntax model



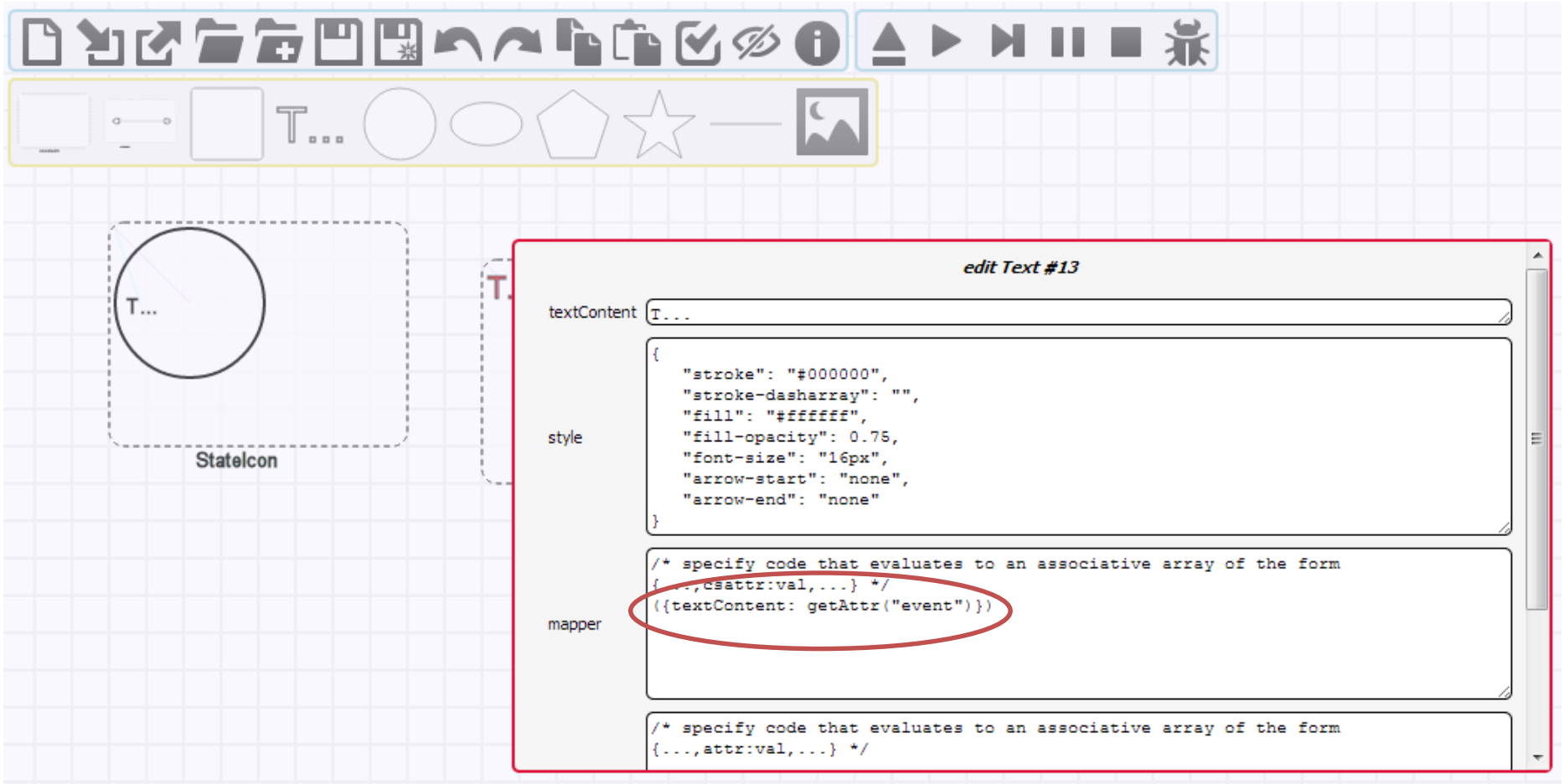
Build a concrete syntax

Display the name of the state



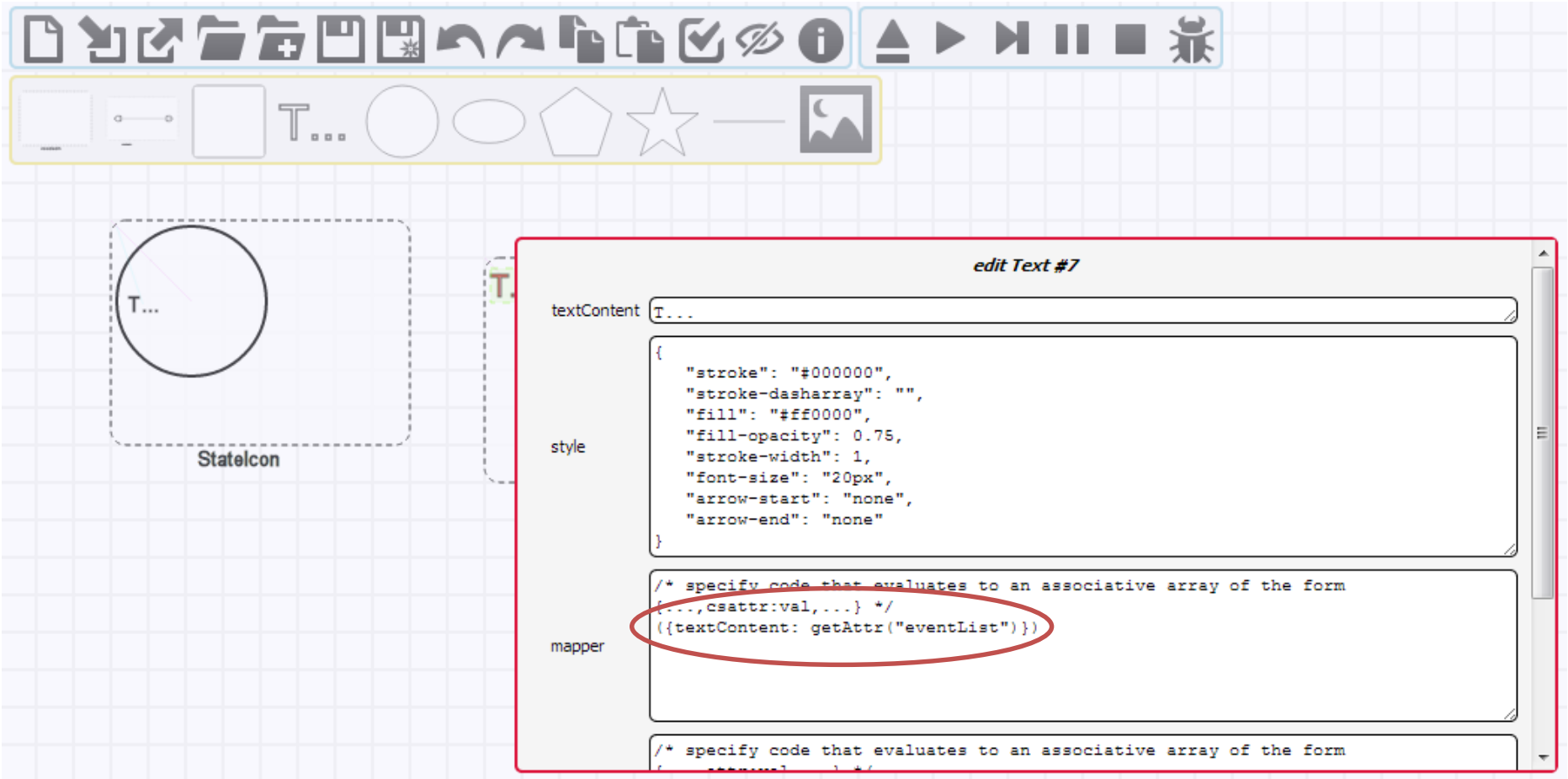
Build a concrete syntax

Display the event on the transition



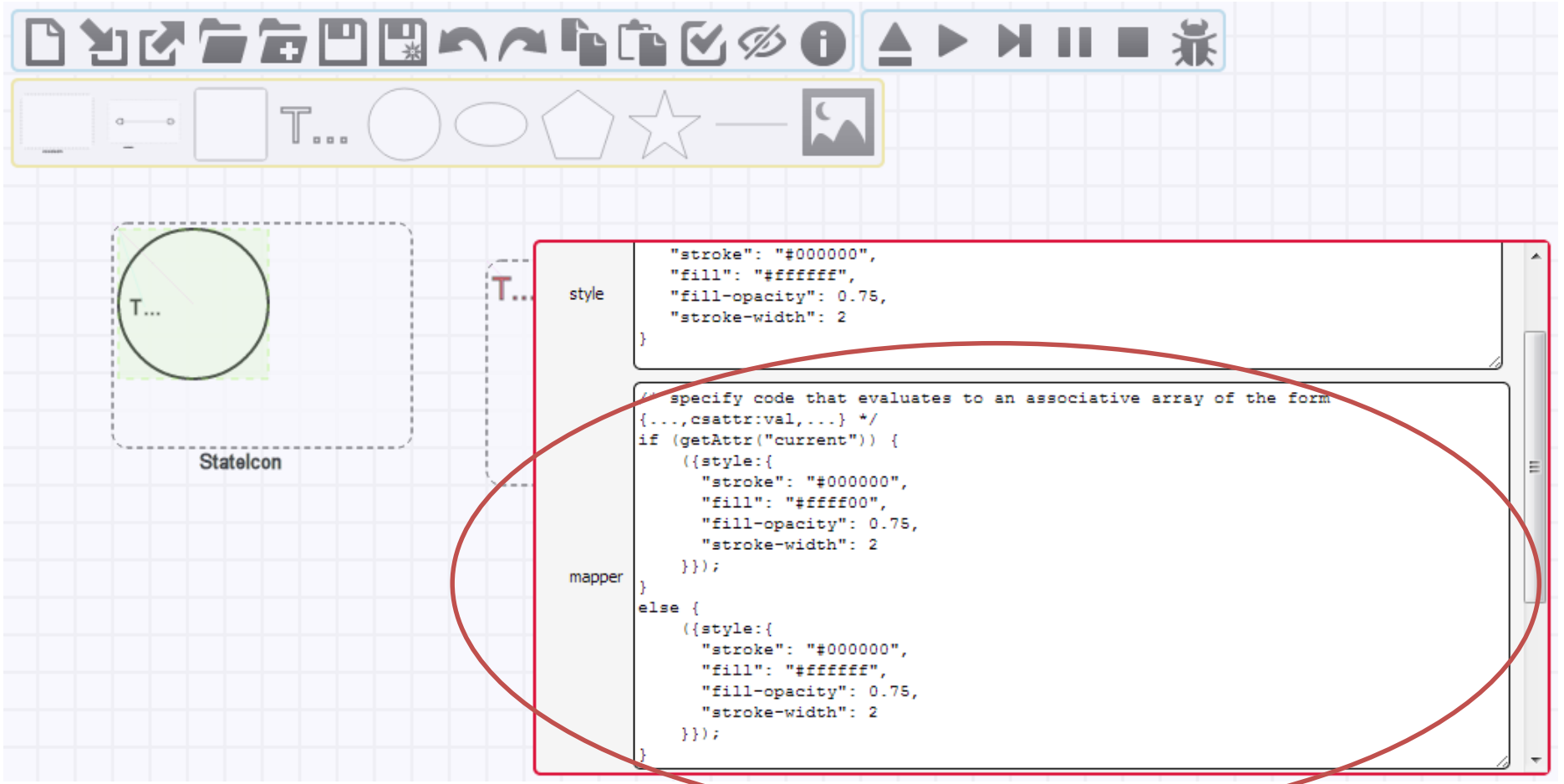
Build a concrete syntax

Display the event list



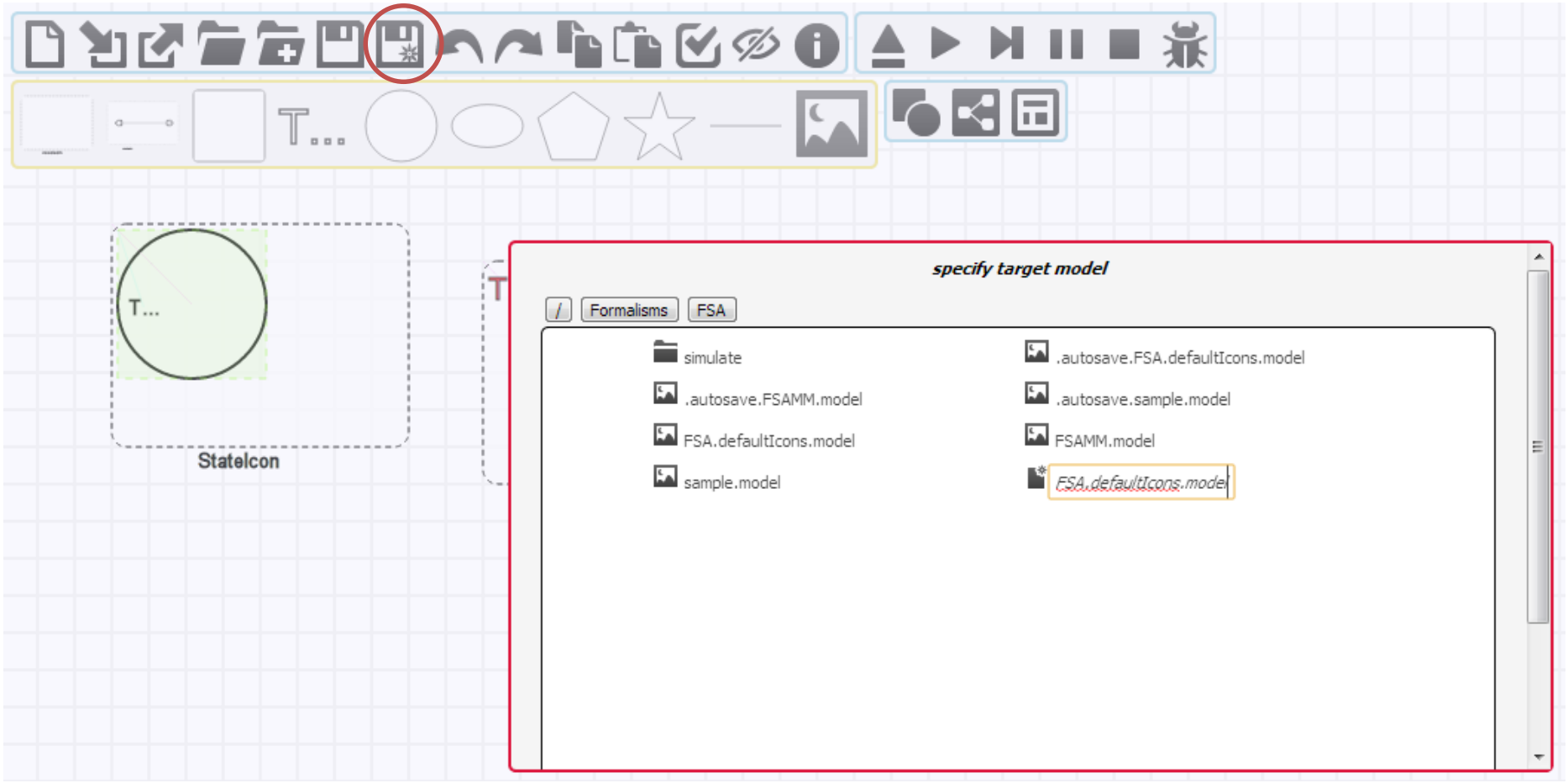
Build a concrete syntax

Change the color of the current state to yellow



Build a concrete syntax

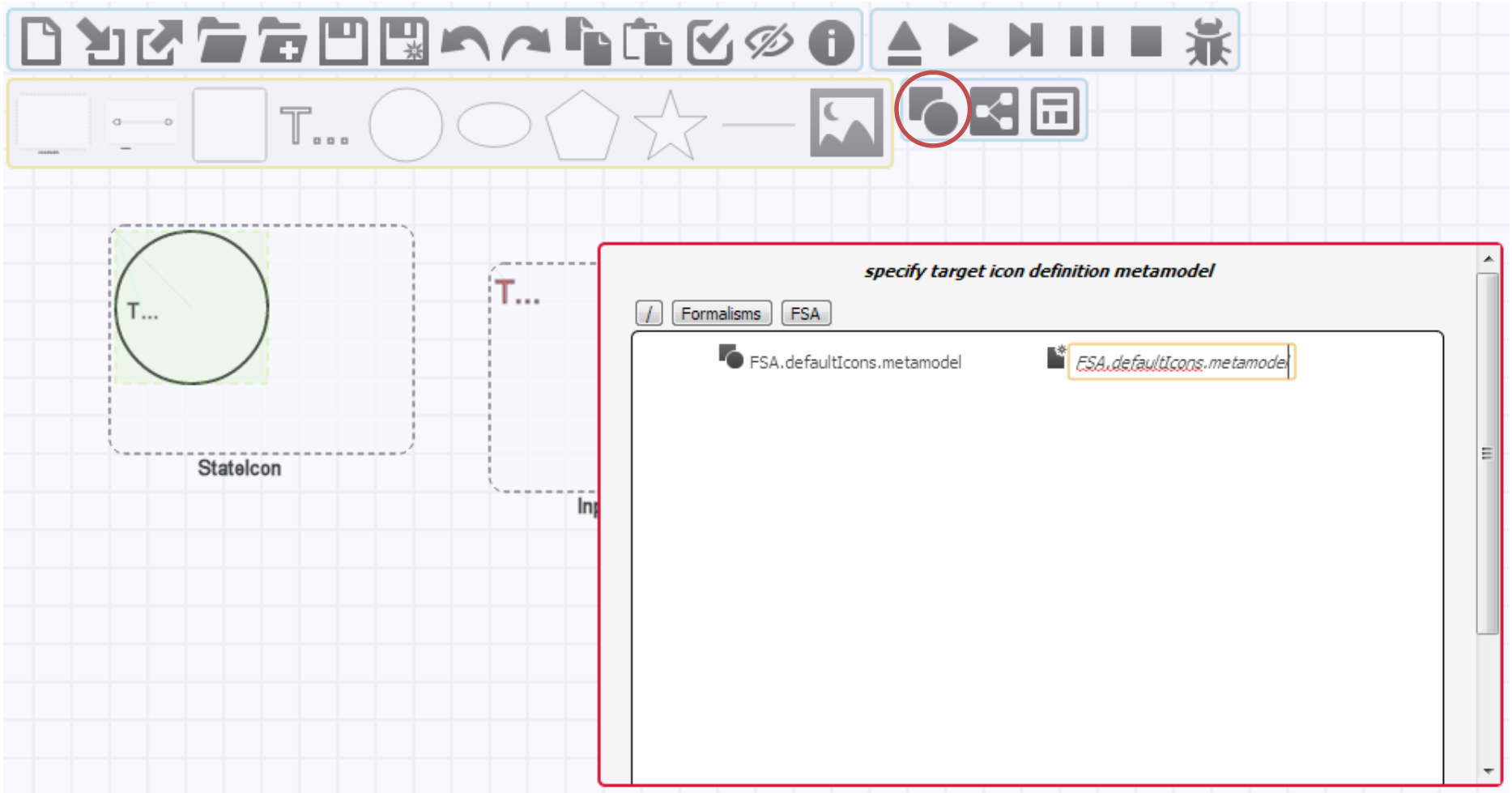
Save as the concrete syntax



The file name must be `METAMODELNAME+“.”+“CONCRETESYNTAXNAME”+“Icons.model”`

Build a concrete syntax

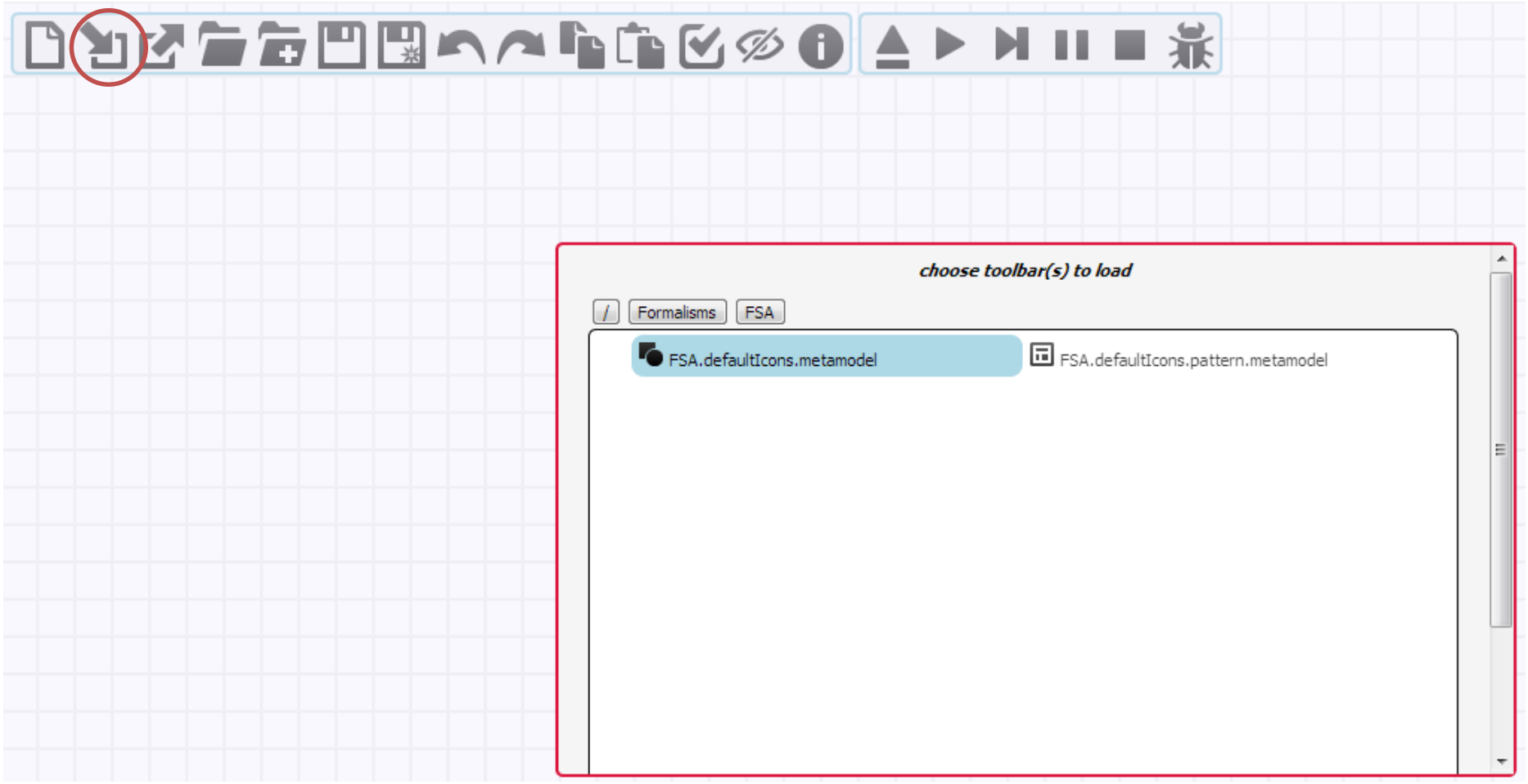
Generate a modeling environment



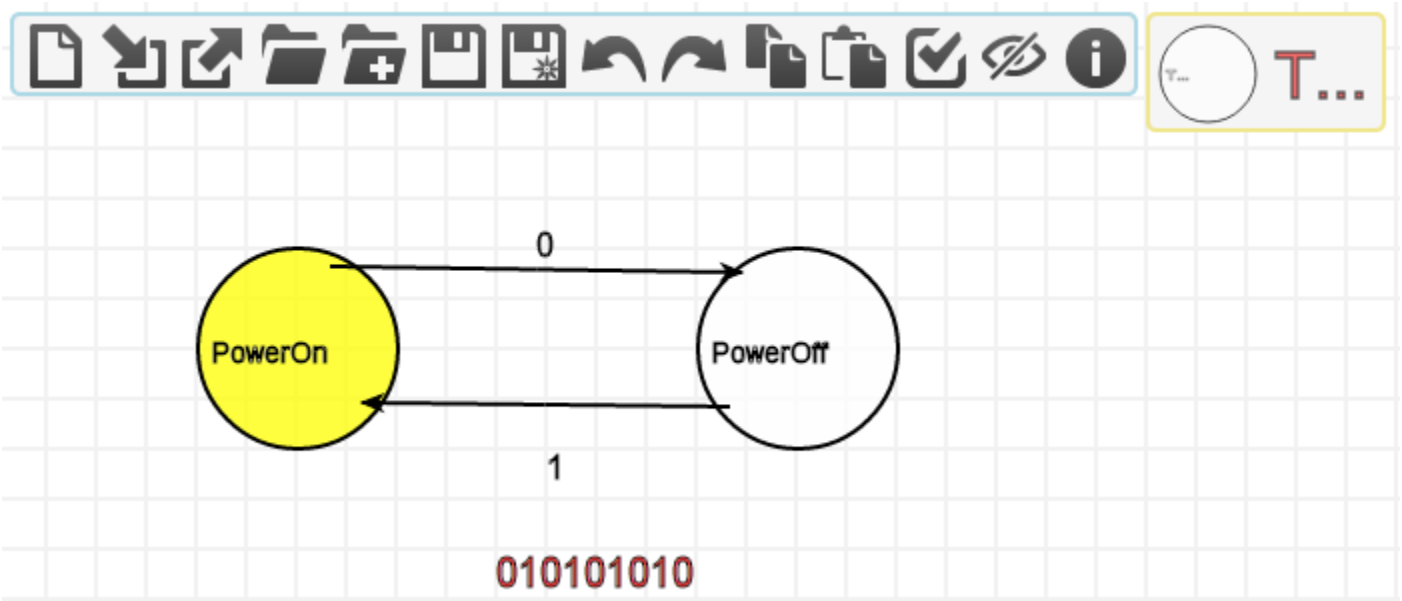
The file name must be `METAMODELNAME+“.”+“CONCRETESYNTAXNAME”+“Icons.metamodel”`

Build a model

Load the FSA formalism toolbar



Build a model



The file name must be `NAME+“.”+“model”`

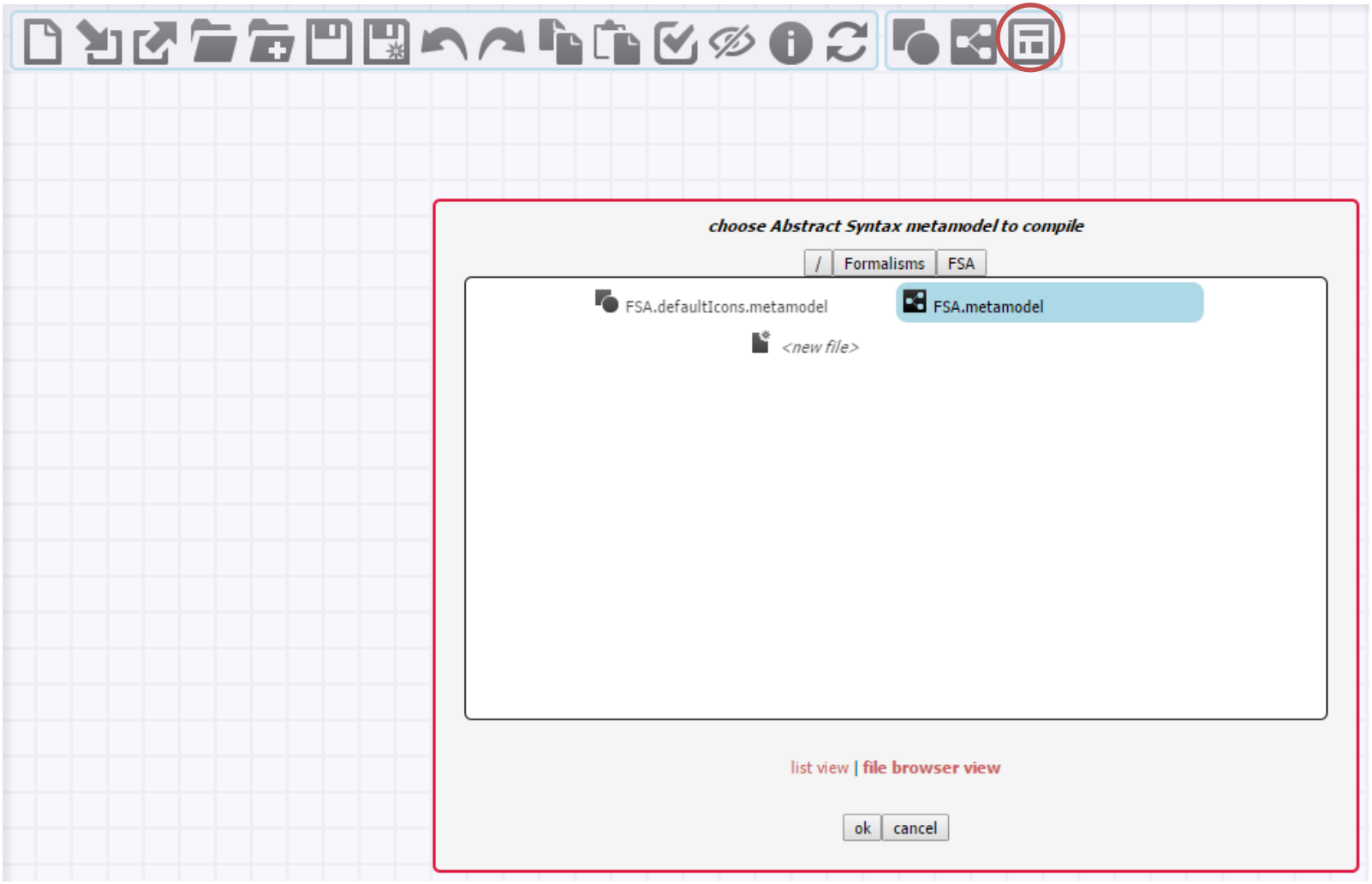
Behavior: Simulation

- Now that we have defined the syntax of the language, we shall define its behavior
- With a model transformation that serves as **simulation**

- So how to do that?
  - Define the rules
  - Define the scheduling of the rules (control flow)

Build a rule

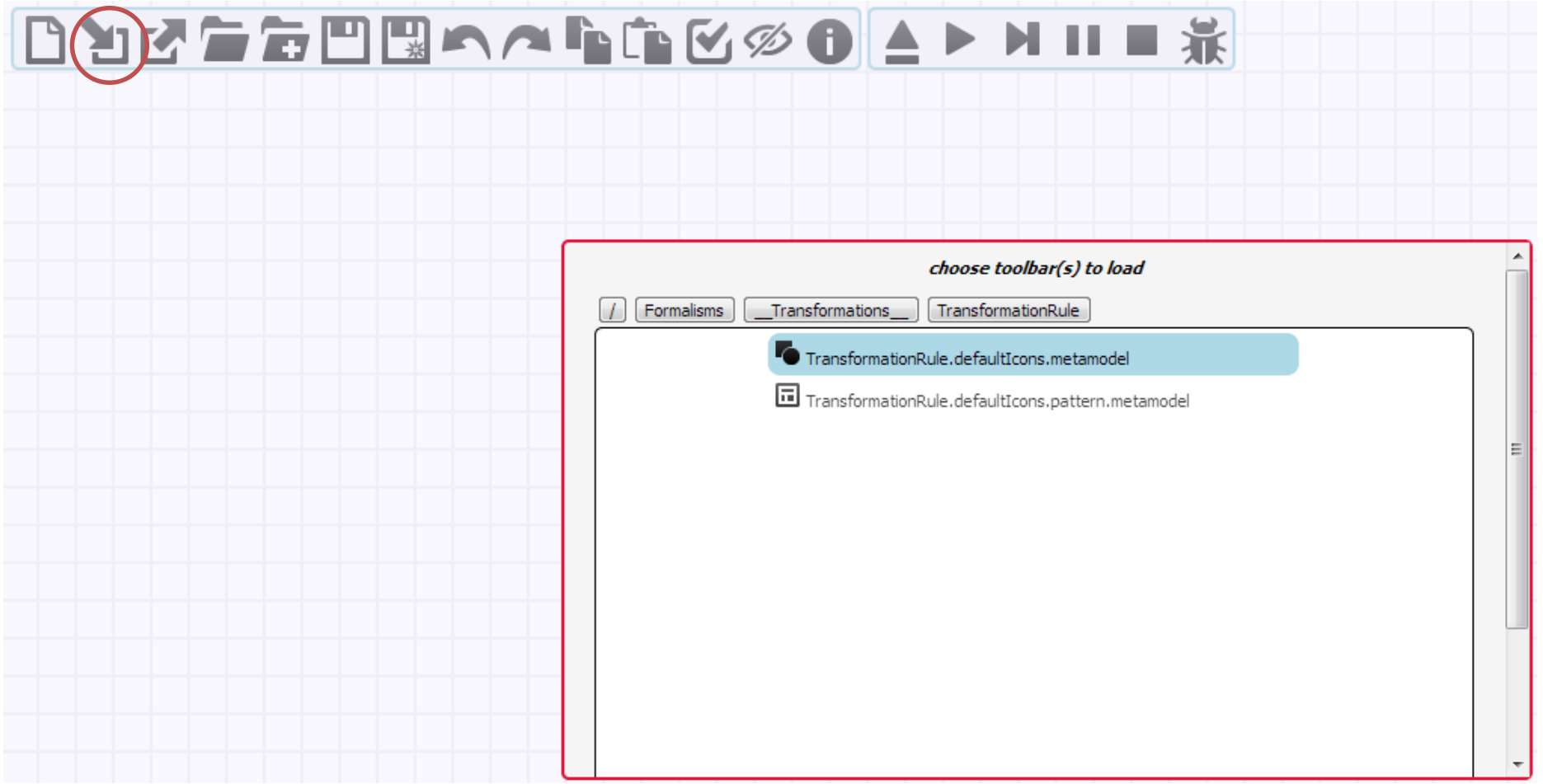
Generate a modeling environment for designing patterns of FSA



Select the file NAME+“.metamodel”

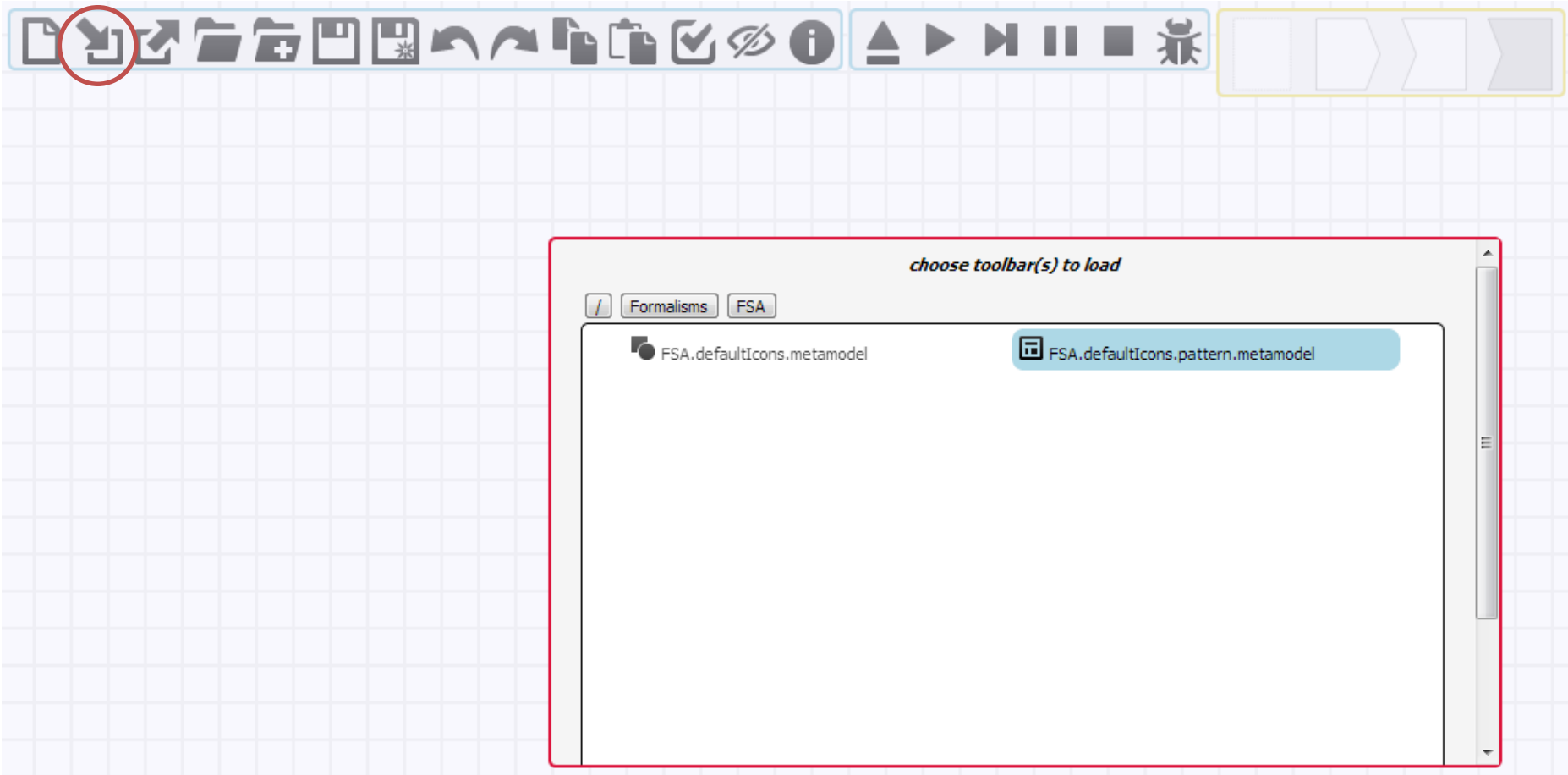
Build a rule

Load the `TransformationRule` formalism toolbar



Build a rule

Load the `FSA.pattern` formalism toolbar

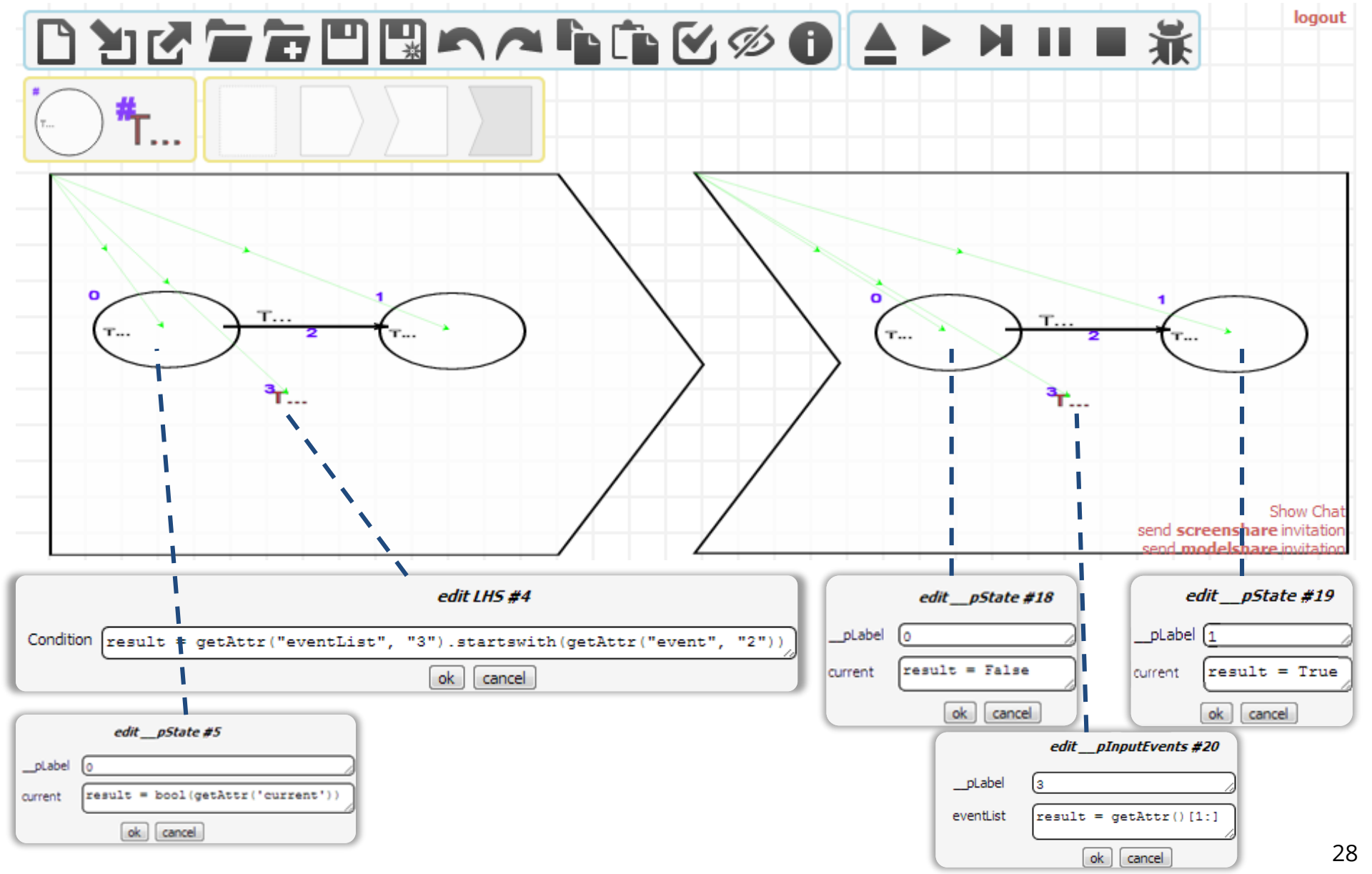


Define the rule

- In this simple FSA example, we can assume that
  - An event is specified by one character
  - There is exactly one state that is current
  - This is a deterministic FSA so it is a DFA
- We can therefore define the behavior of such FSAs with only one rule:

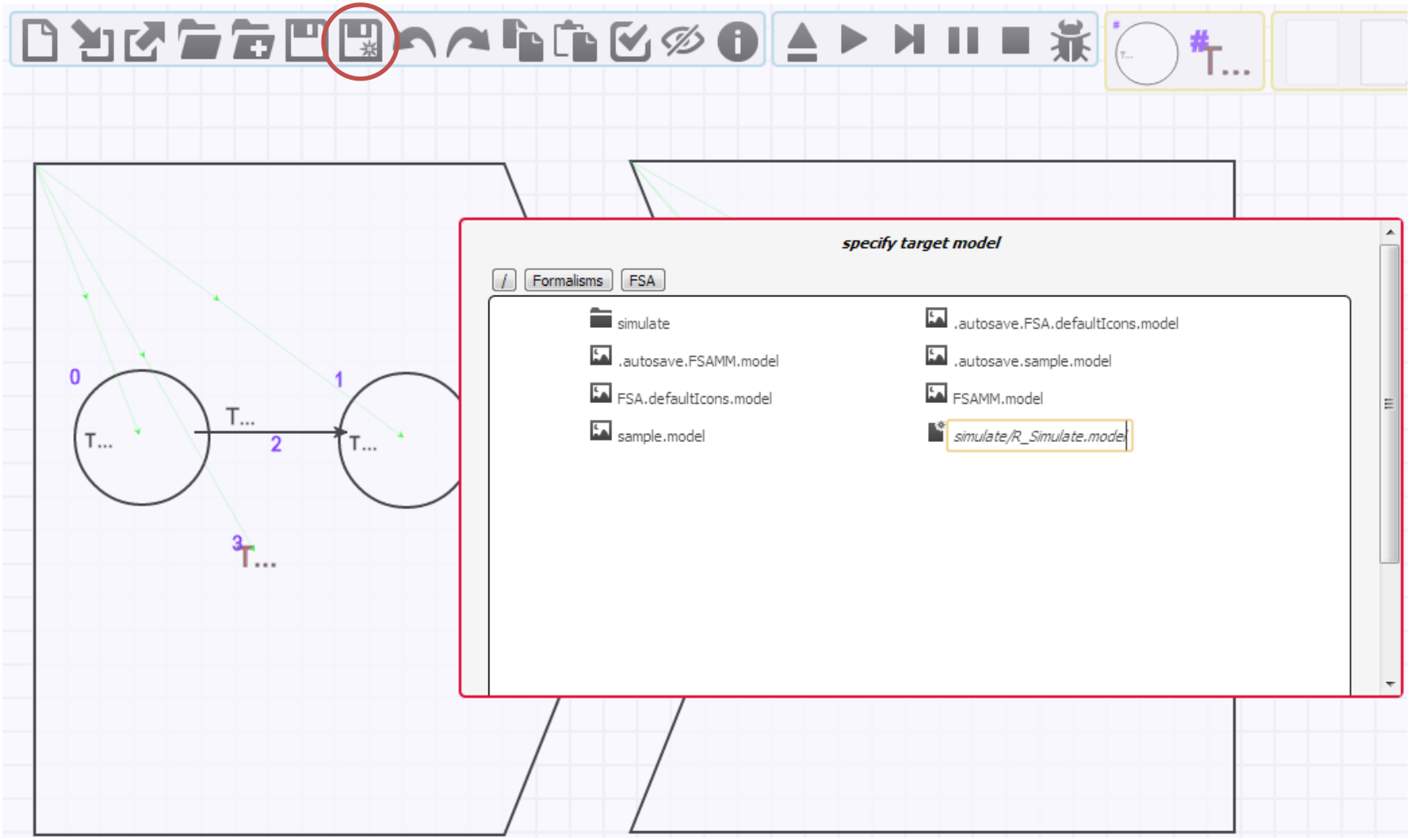
If there are two states connected by a transition such that
the event on the transition is equal to
the first input in the event list
and the incoming state is the current state,
then remove the first event from the event list
and set the current state to be the outgoing state only

As a graph transformation rule



Build the rule

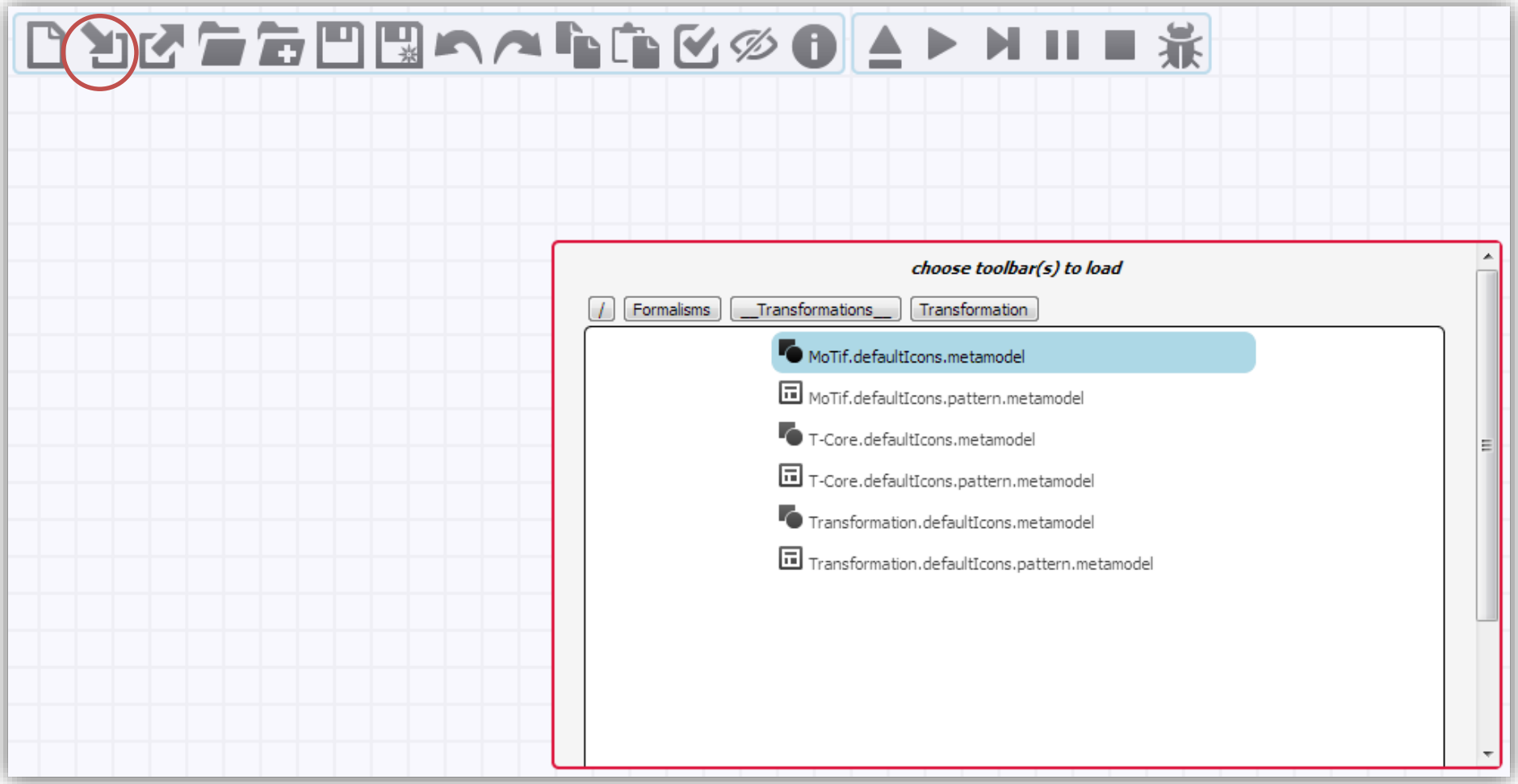
Save as the transformation rule



The file name must be “R_”+NAME+“.model”

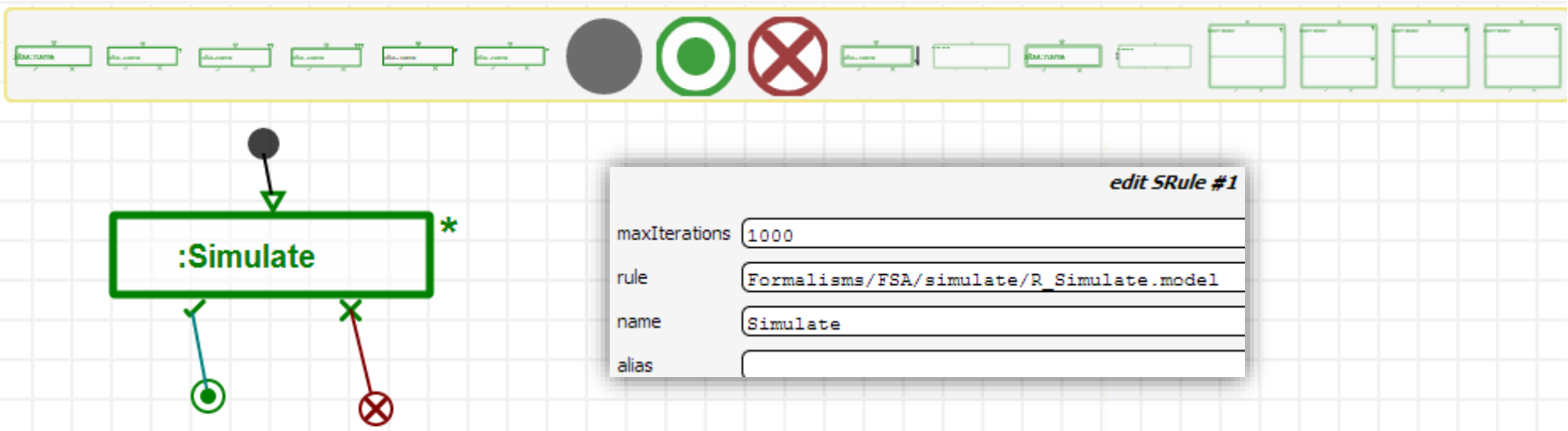
Build the rule scheduling

Load the `MoTif` formalism toolbar

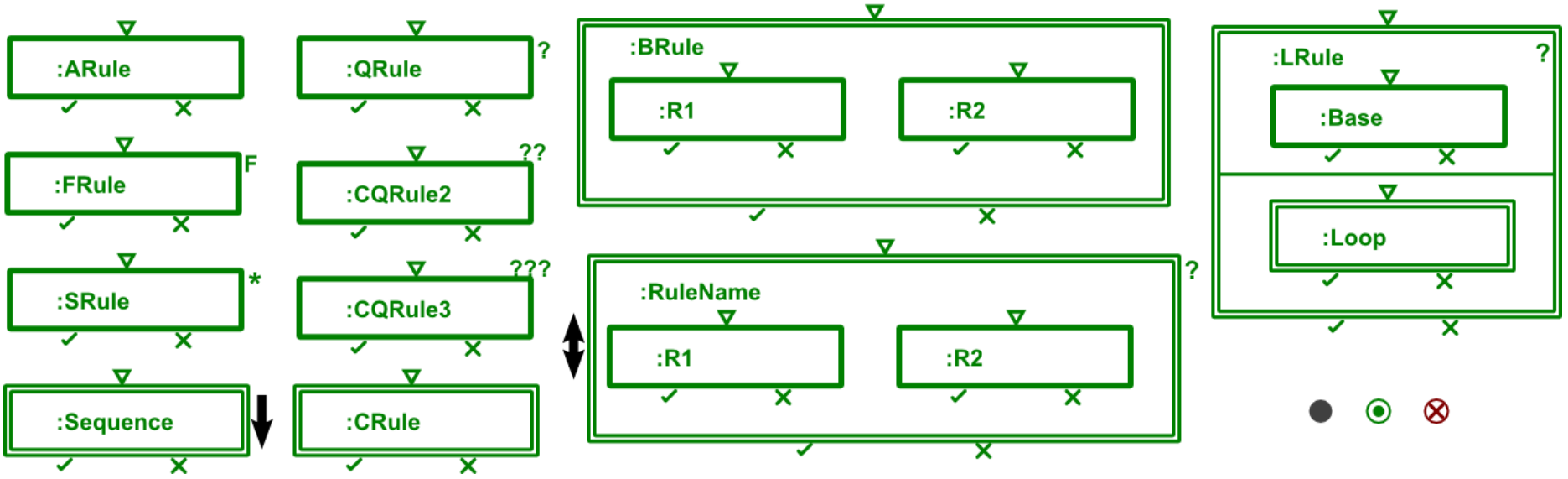


Rule scheduling

- Using the MoTif language, we can execute our `simulate` rule as an `SRule`
- `SRule`: Apply the rule recursively as long as possible
  - Find a match, rewrite the model, then re-match, rewrite the model, etc.

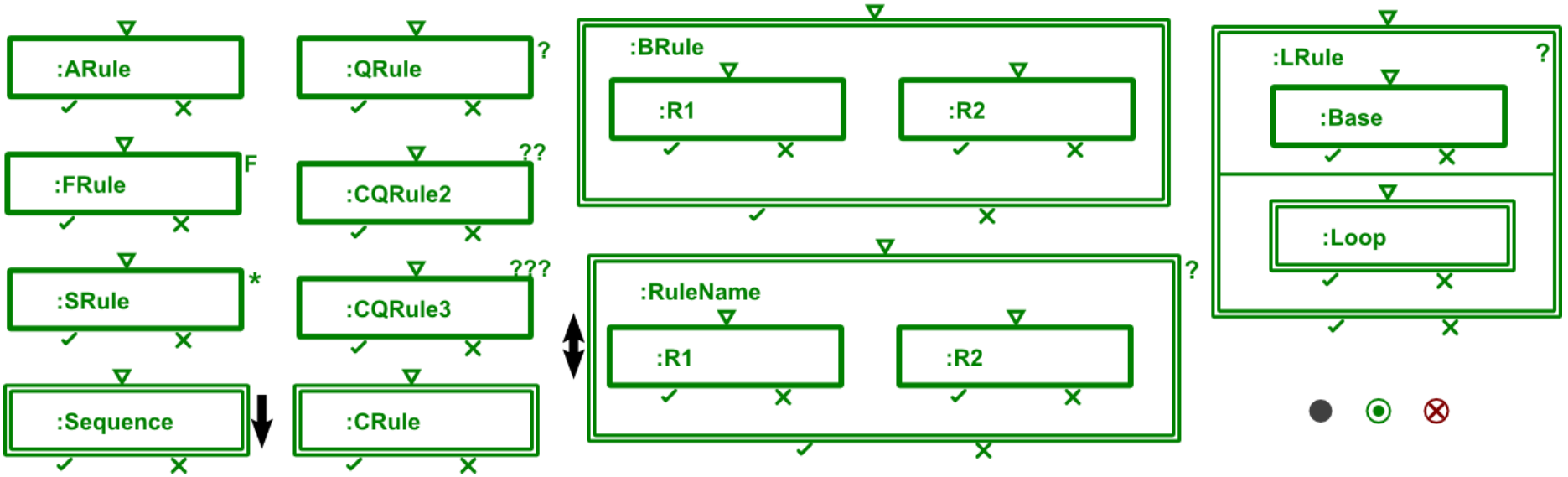


MoTif constructs



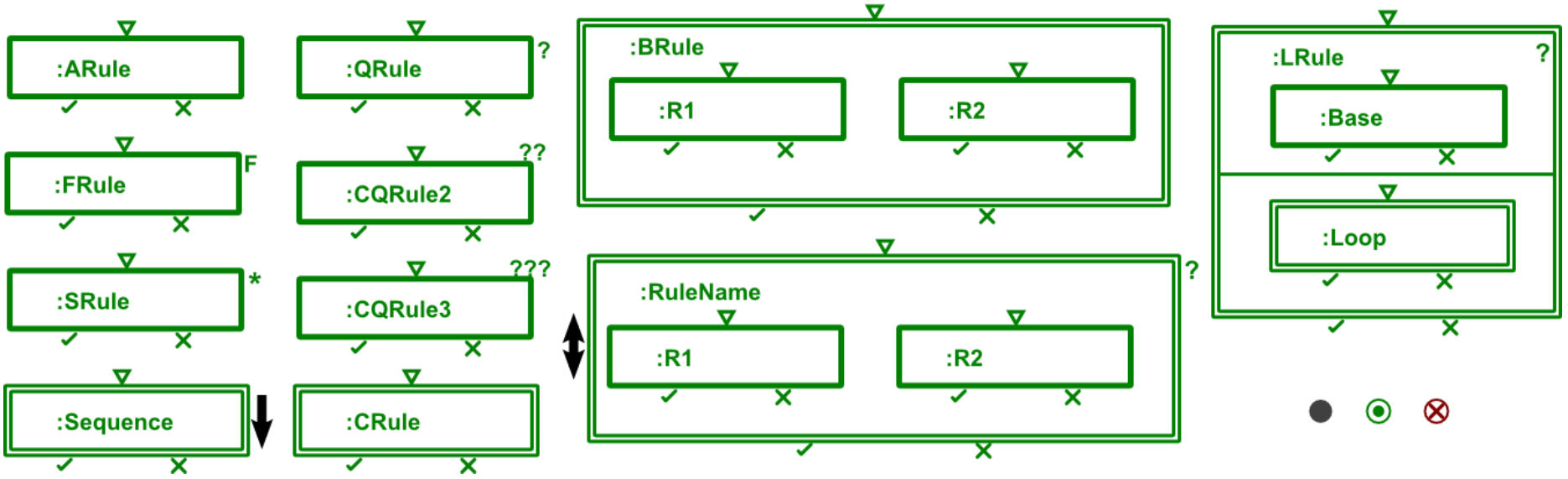
- `ARule`: (atomic) Applies the rule on one match
- `FRule`: (for all) Applies the rule on all matches found in parallel
- `SRule`: (star) Applies the rule recursively as long as matches can be found

MoTif constructs



- `QRule`: (query) Finds a match for the LHS
- `CQRule2,3`: (composite queries) Nested query for 2 or 3 levels
- `Sequence`: Applies a set of rules in ordered sequence
- `CRule`: (composite) Refers to another (sub-)transformation

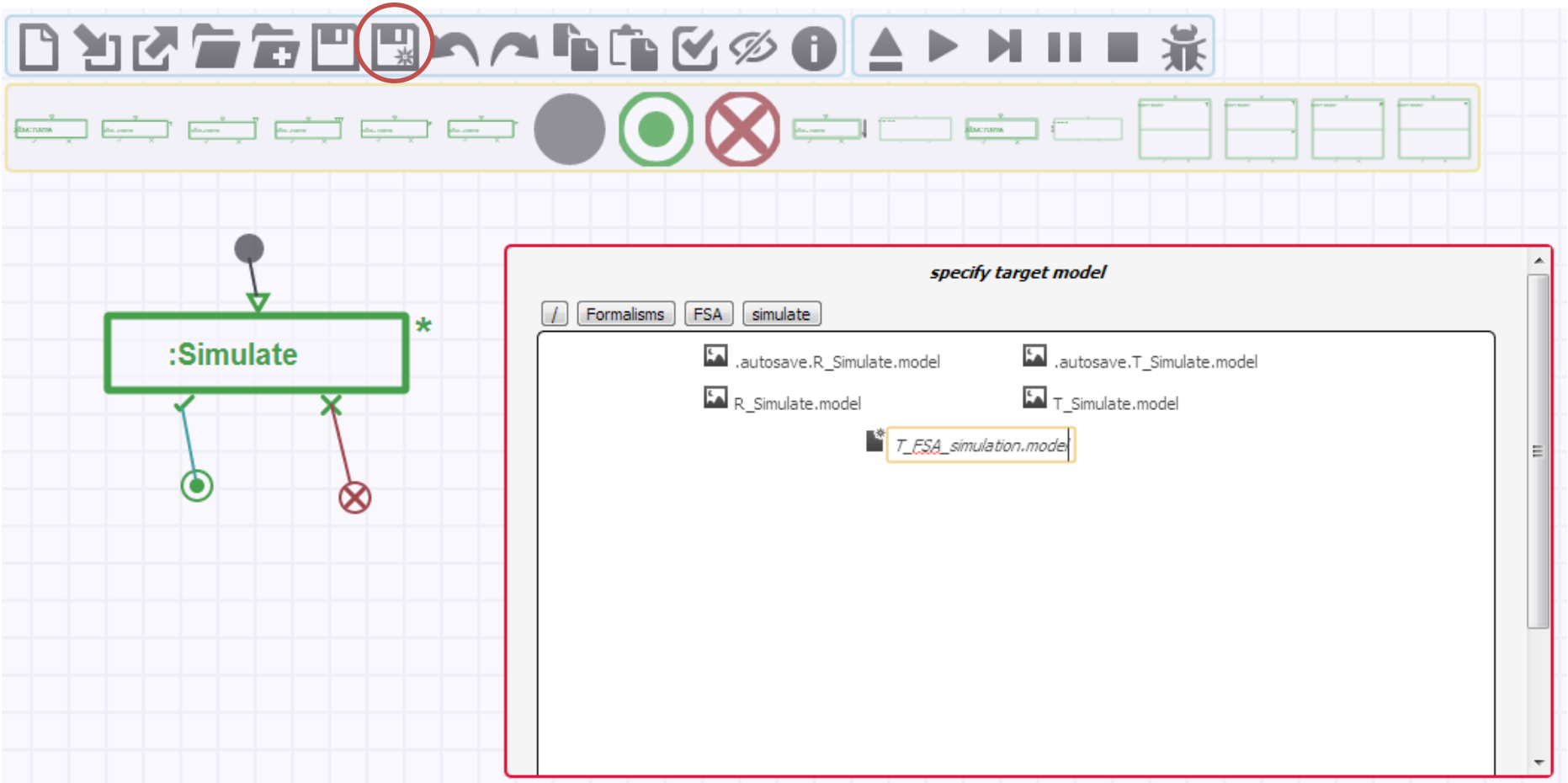
MoTif constructs



- `BRule`: (branch) non-deterministically selects one successful branch of execution
- `BSRule`: (branch star) Recursive `BRule`
- `LRule`: (loop) For each match of the base rule, apply the loop rule
- `Start`, `EndSuccess`, `EndFail` pseudo-states

Build the rule scheduling

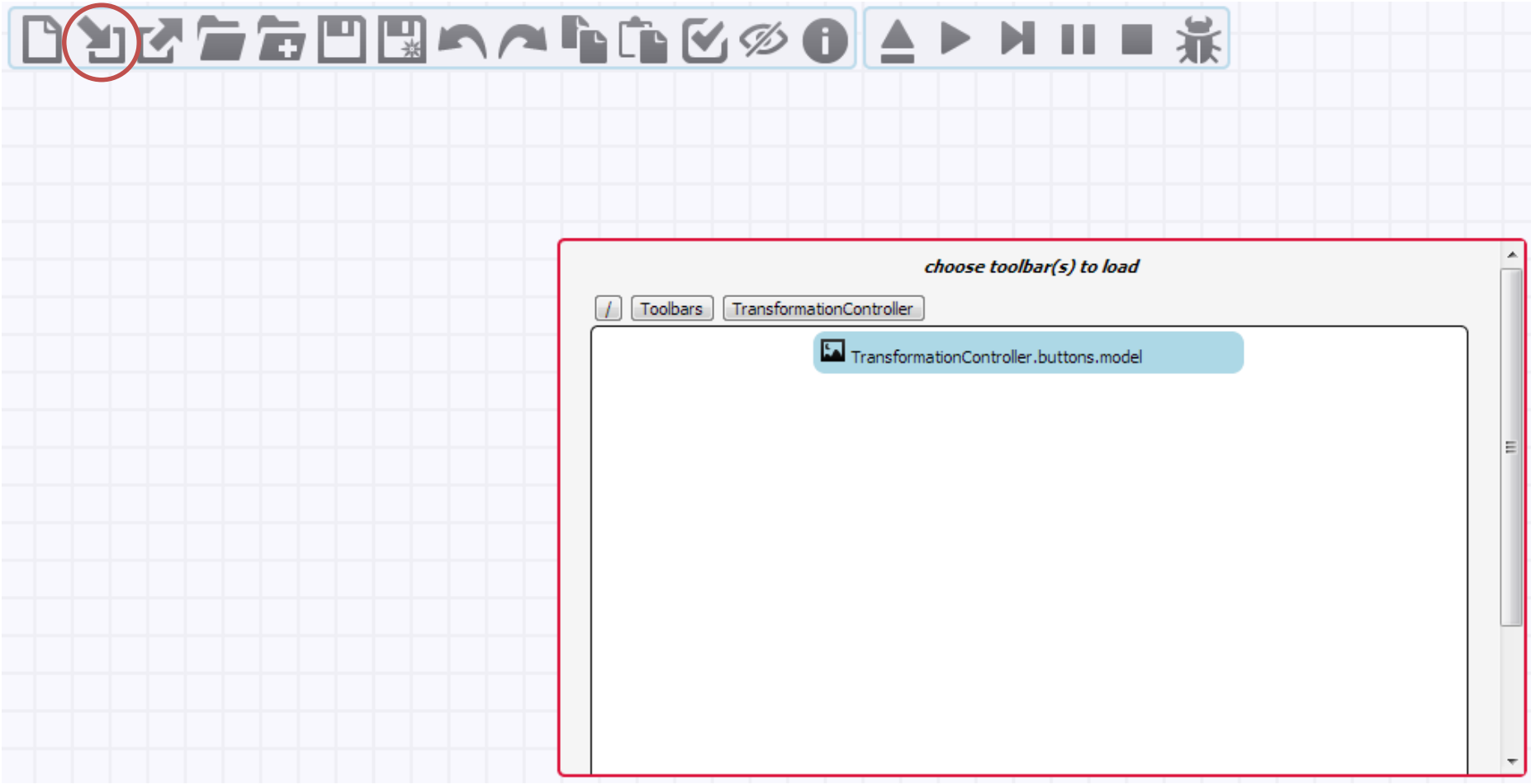
Save as the MoTif model



The file name must be “T_”+NAME+“.model”

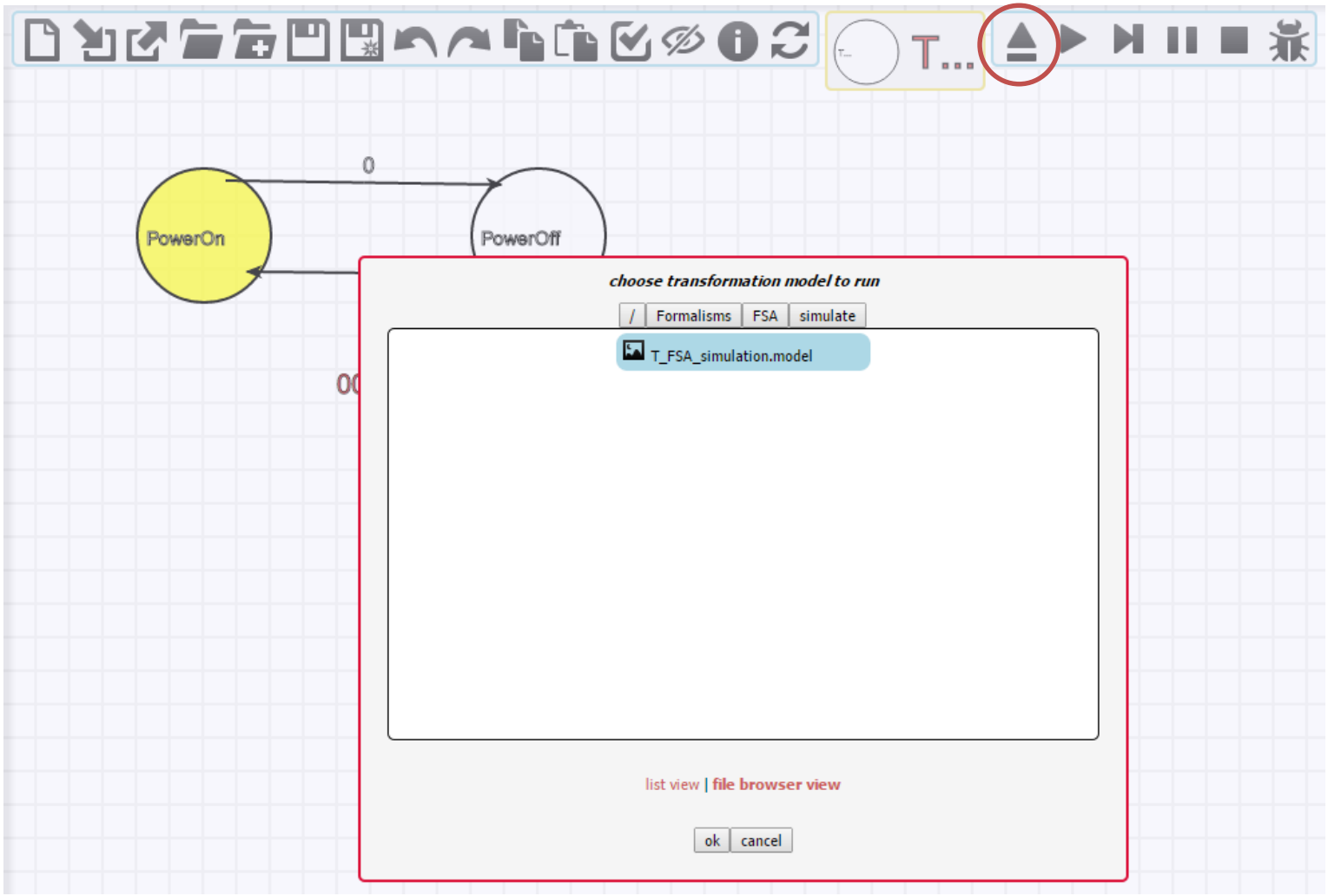
Execute the model transformation

Load the `TransformationController` toolbar



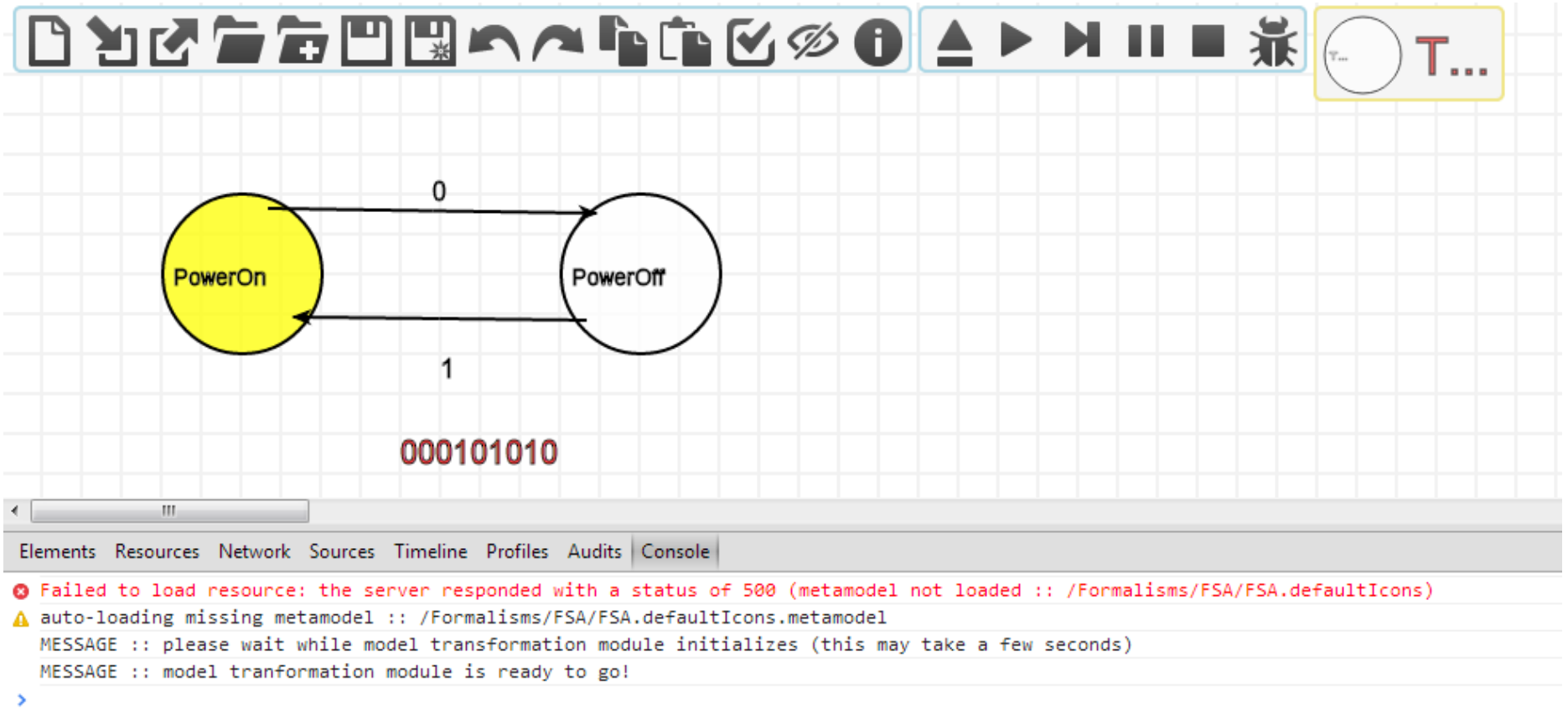
Execute the model transformation

Build/open an FSA model & load the `T_FSA_simulation` transformation



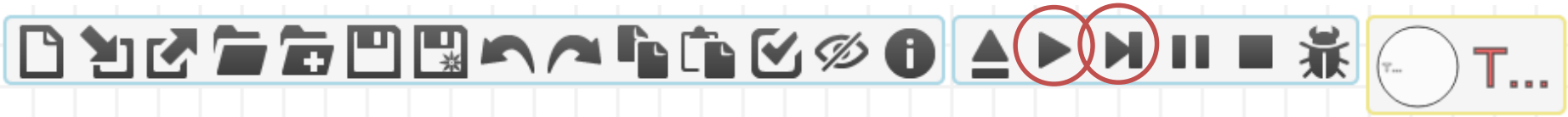
Execute the model transformation

Press <SHIFT>+<CTRL>+i to view the chrome console

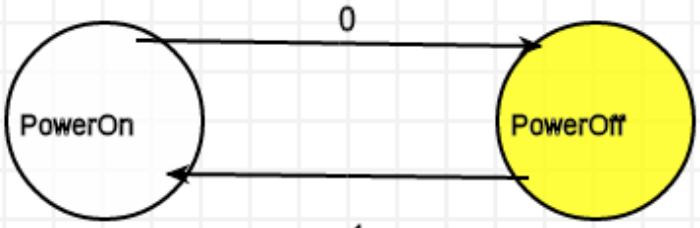


Execute the model transformation

Execute the transformation



Continuously or step-by-step



00101010

Elements Resources Network Sources Timeline Profiles Audits Console

```
Failed to load resource: the server responded with a status of 500 (metamodel not loaded :: /Formalisms/FSA/FSA.defaultIcons)
auto-loading missing metamodel :: /Formalisms/FSA/FSA.defaultIcons.metamodel
MESSAGE :: please wait while model transformation module initializes (this may take a few seconds)
MESSAGE :: model tranformation module is ready to go!
MESSAGE :: rule succeeded (:Simulate)
MESSAGE :: transformation(s) terminated with status :: Success in 1.05216988037
MESSAGE :: transformation stopped
```