
TTool Training

III. Using TTool

Ludovic Apvrille

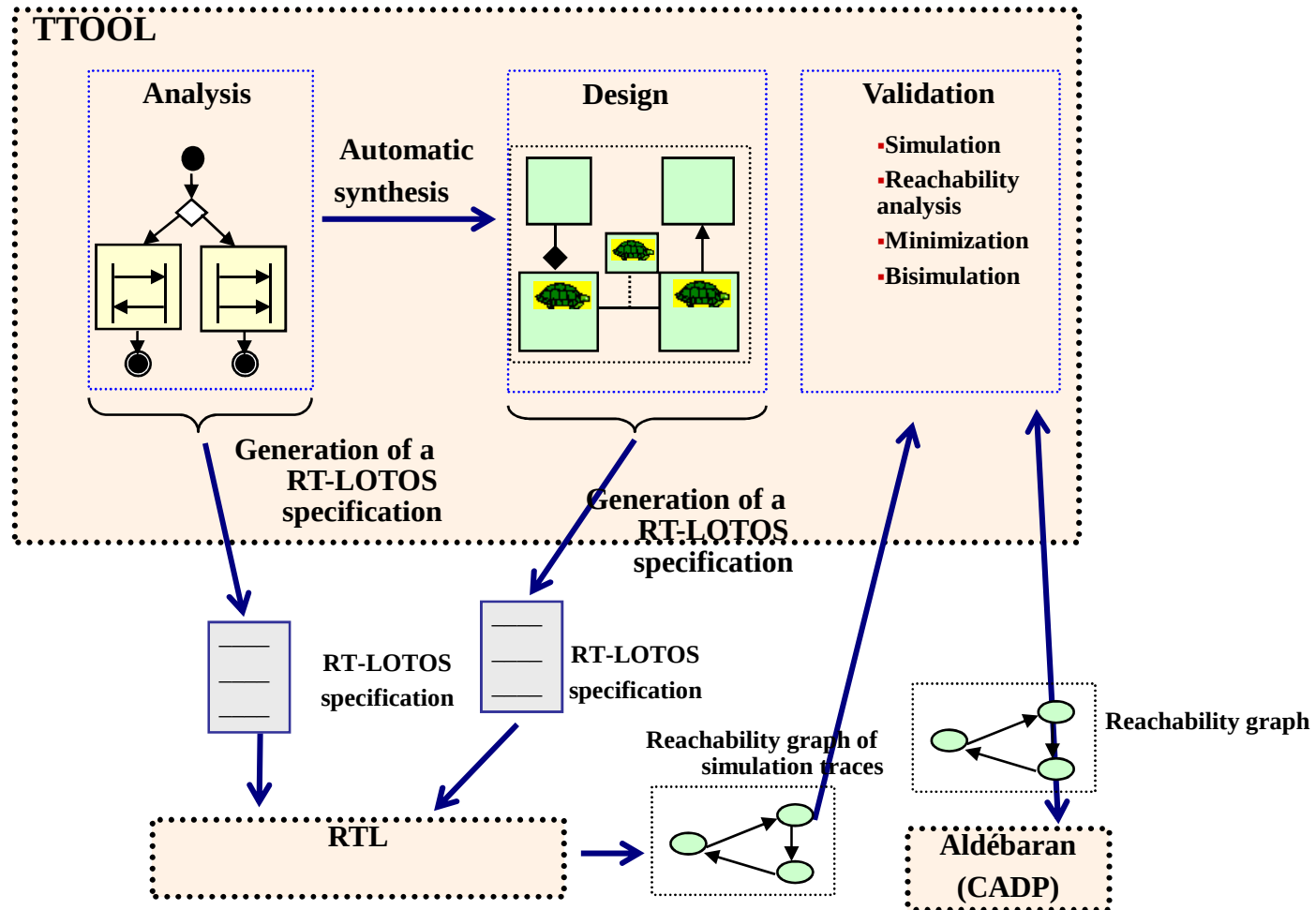
ludovic.apvrille@telecom-paris.fr

Eurecom, Office 223

I. Introduction

- ➔  **TTool: main features**
-  **Installing TTool**
-  **Diagramming with TTool**
-  **Formal validation**
-  **Java code generation**

TTool: General Overview



I. Introduction

 **TTool: main features**

  **Installing TTool**

 **Diagramming with TTool**

 **Formal validation**

 **Java code generation**

Inside a TTool package

 **ttool.jar**

 **launcher.jar**

 **config.xml**

 **ttoolmanual.doc**

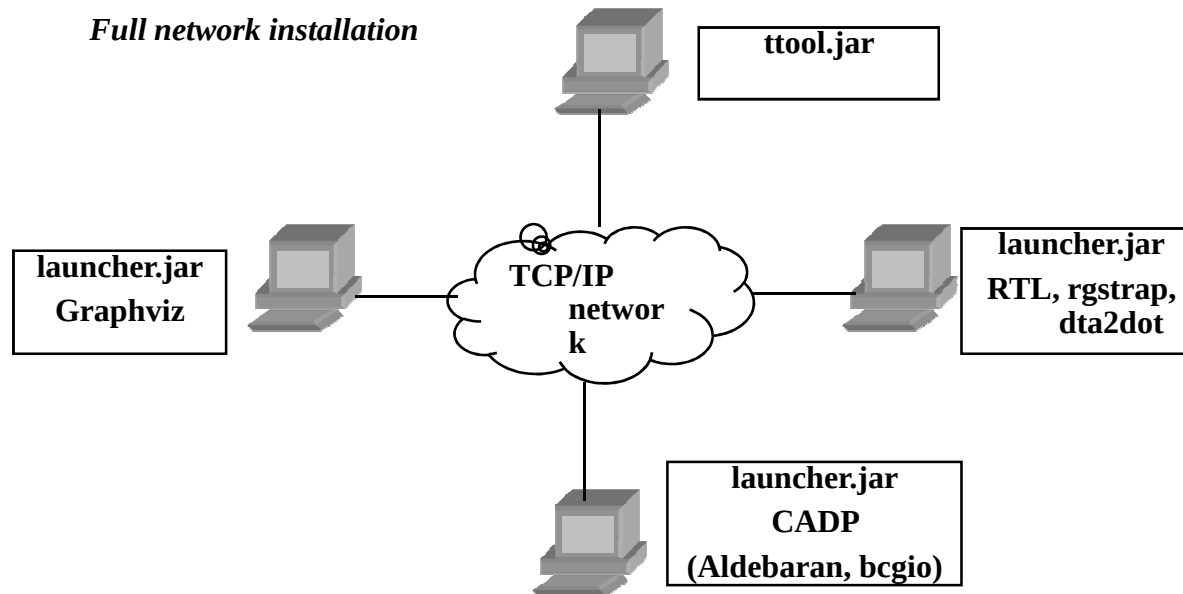
 ***.xml: examples (files into which ttool modeling are saved)**

 ***.lib: librairies (diagrams that can be included into other diagrams)**

Starting TTool

- ❏ JVM 1.4.2 or later must be installed on every machine running TTool or an external program
- ❏ Configuration of the xml file
 - ❑ *See next slide*
- ❏ Starting of the main program from a terminal
 - ❑ `java -Xmx256m -jar ttool.jar MyConfig.xml`
- ❏ The launcher must be started on each machine on which an external application is used (RTL, CADP/Aldébaran, Graphviz)
 - ❑ `java -jar launcher.jar`

Example of Configuration



Configuring config.xml

```
<RTLHost data="cow13.eurecom.fr" />
<RTLPath data="/homes/apvrille/RT-LOTOS/Linux/rtl/bin-linux/rtl" />
<DTA2DOTPath data="/homes/apvrille/RT-LOTOS/Linux/rtl/bin-linux/dta2dot" />
<RG2TLSAPath data="/homes/apvrille/RT-LOTOS/Linux/rtl/bin-linux/rg2tlsa" />
<RGSTRAPPath data="/homes/apvrille/RT-LOTOS/Linux/rtl/bin-linux/rgstrap" />
<DOTTYHost data="chaland.eurecom.fr" />
<DOTTYPath data="/homes/apvrille/softs/gv1.7c/bin/dotty" />
<AldebaranHost data="cow13.eurecom.fr" />
<AldebaranPath data="/homes/apvrille/cadp/bin.iX86/aldebaran" />
<BcgioPath data="/homes/apvrille/cadp/bin.iX86/bcg_io" />
<FILEPath data="U:\RT-LOTOS\TURTLEModeling" />
<LIBPath data="U:\RT-LOTOS\TURTLELibrary" />
<IMGPath data="U:\Figure" />
<LOTOSPath data="U:\RT-LOTOS\TURTLEModeling" />
<GGraphPath data="U:\RT-LOTOS\TURTLEGraphs" />
<TGraphPath data="U:\RT-LOTOS\TURTLEGraphs" />
<TToolUpdateURL data="http://www.eurecom.fr/~apvrille/TURTLE/ttoolversion.html" />
<TToolUpdateProxy data="false" />
<TToolUpdateProxyPort data="8080" />
<TToolUpdateProxyHost data="To Be Completed" />
```


I. Introduction

 **TTool: main features**

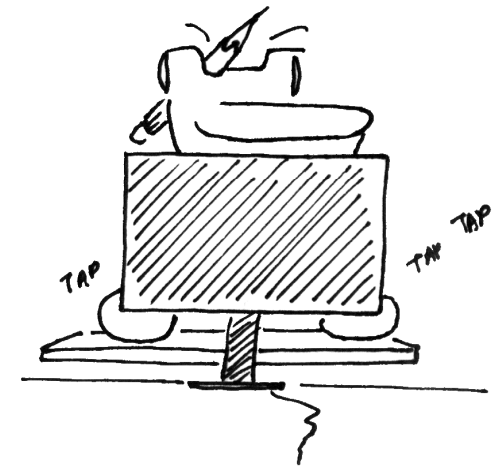
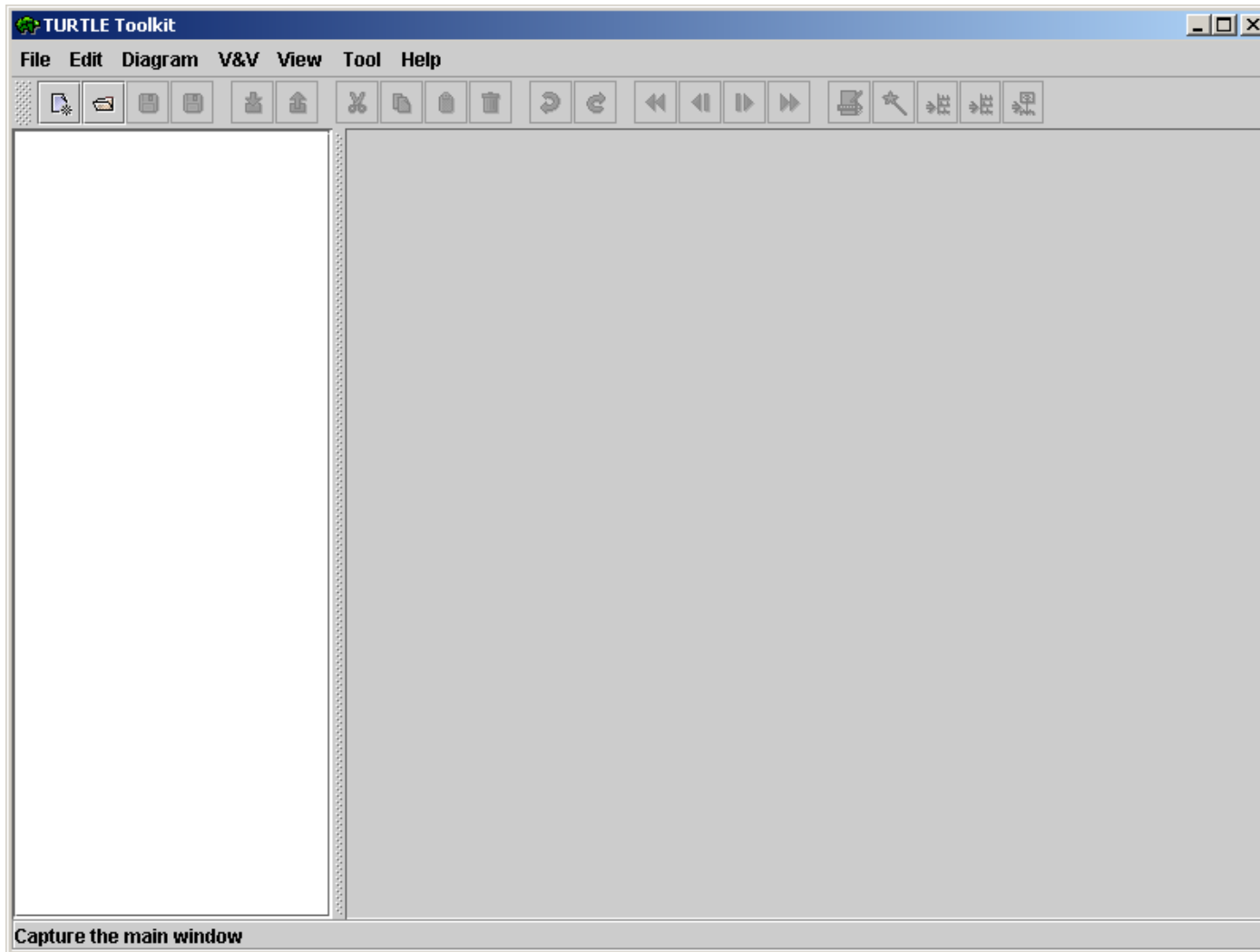
 **Installing TTool**

  **Diagramming with TTool**

 **Formal validation**

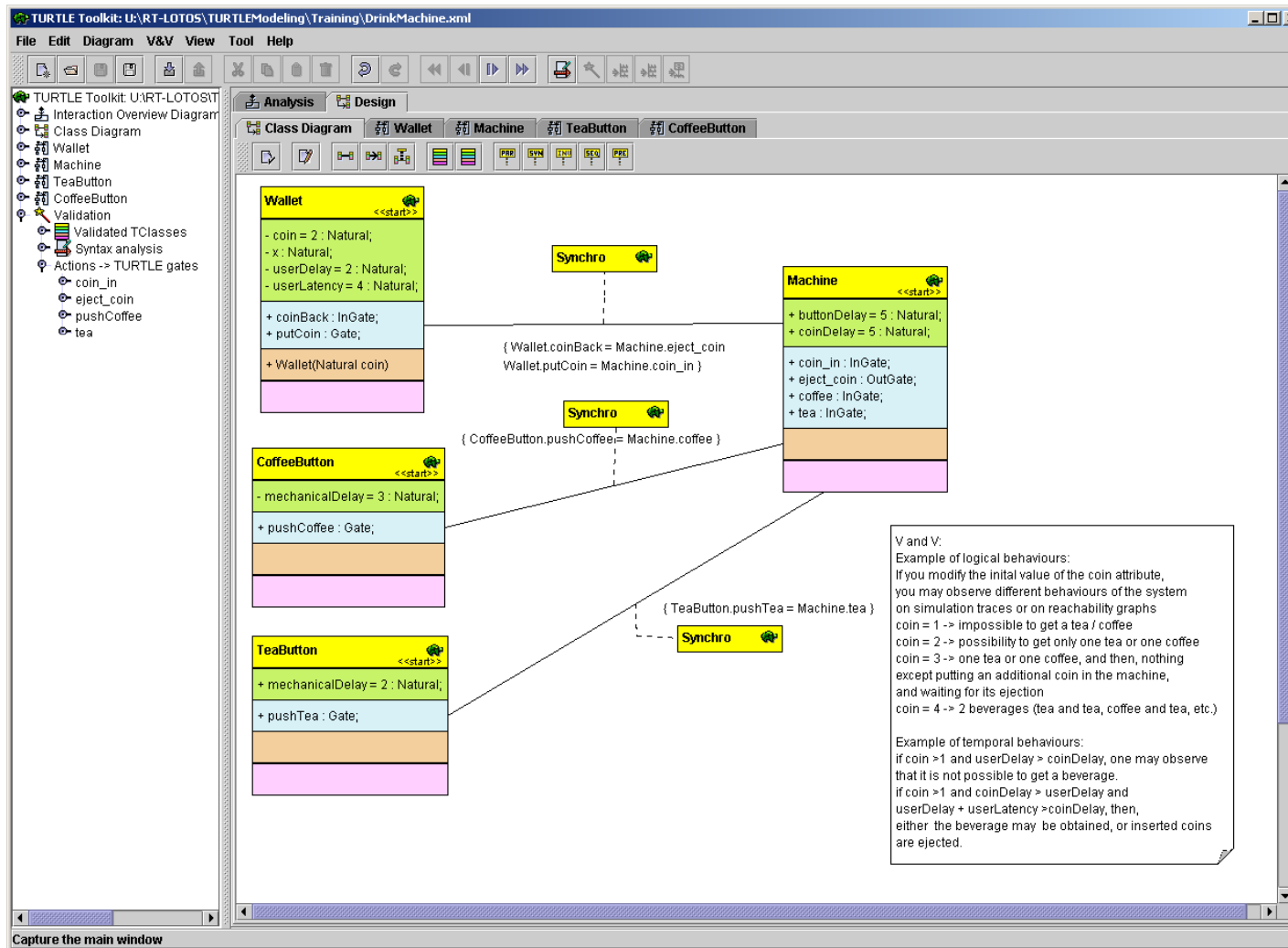
 **Java code generation**

Starting TTool



Opening a TURTLE Modeling

 Menu File, Open, and then select the desired modeling



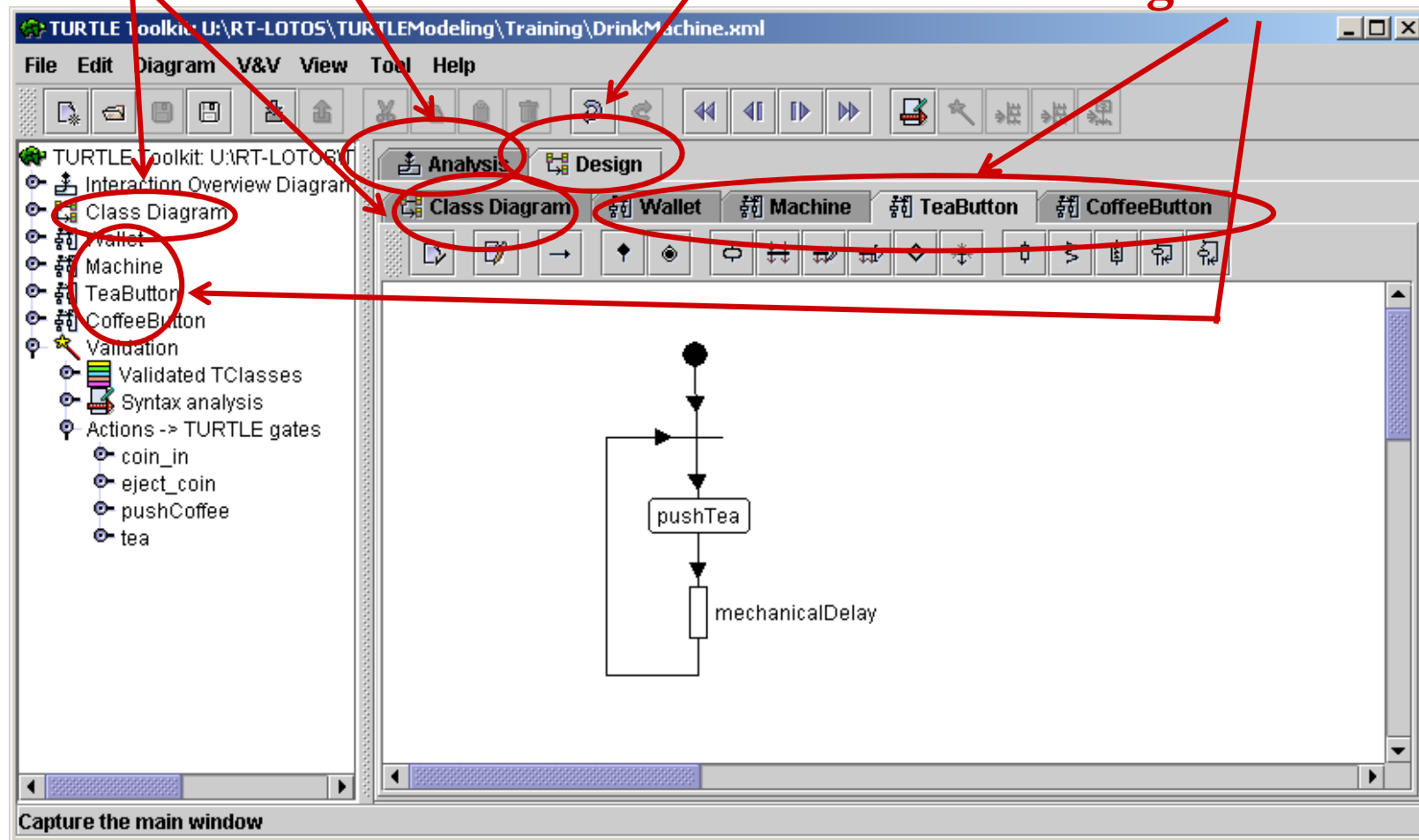
Navigating into Diagrams

**Class
Diagram**

Analysis

Design

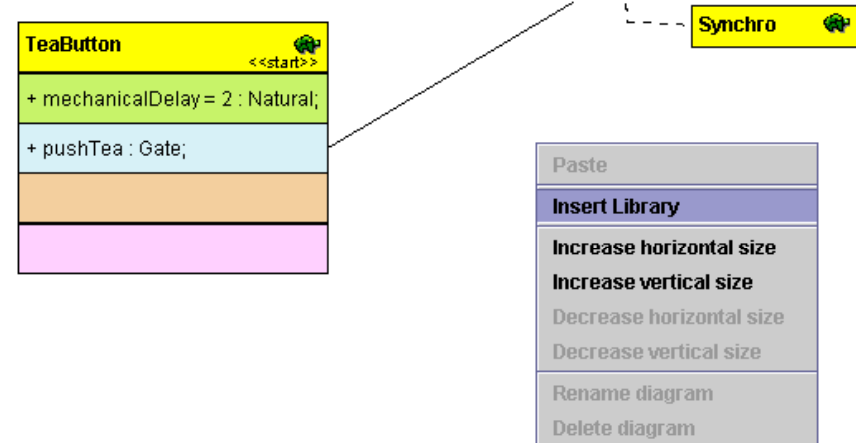
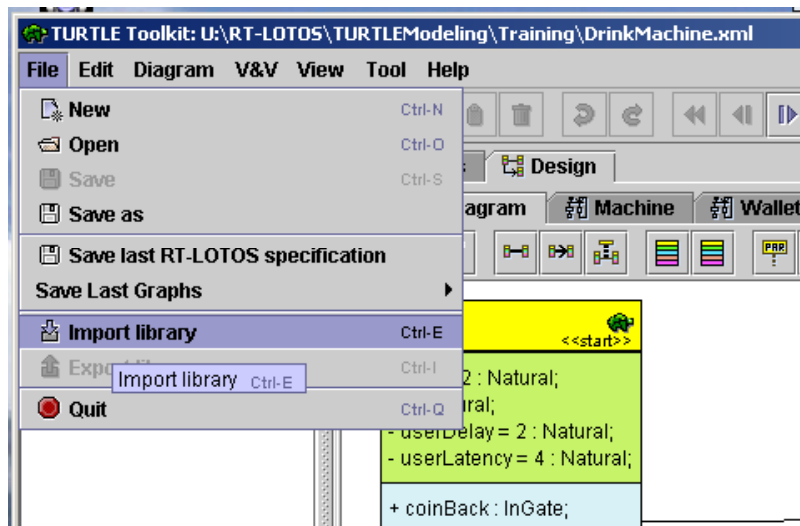
**Activity
Diagrams**



Dealing with Libraries

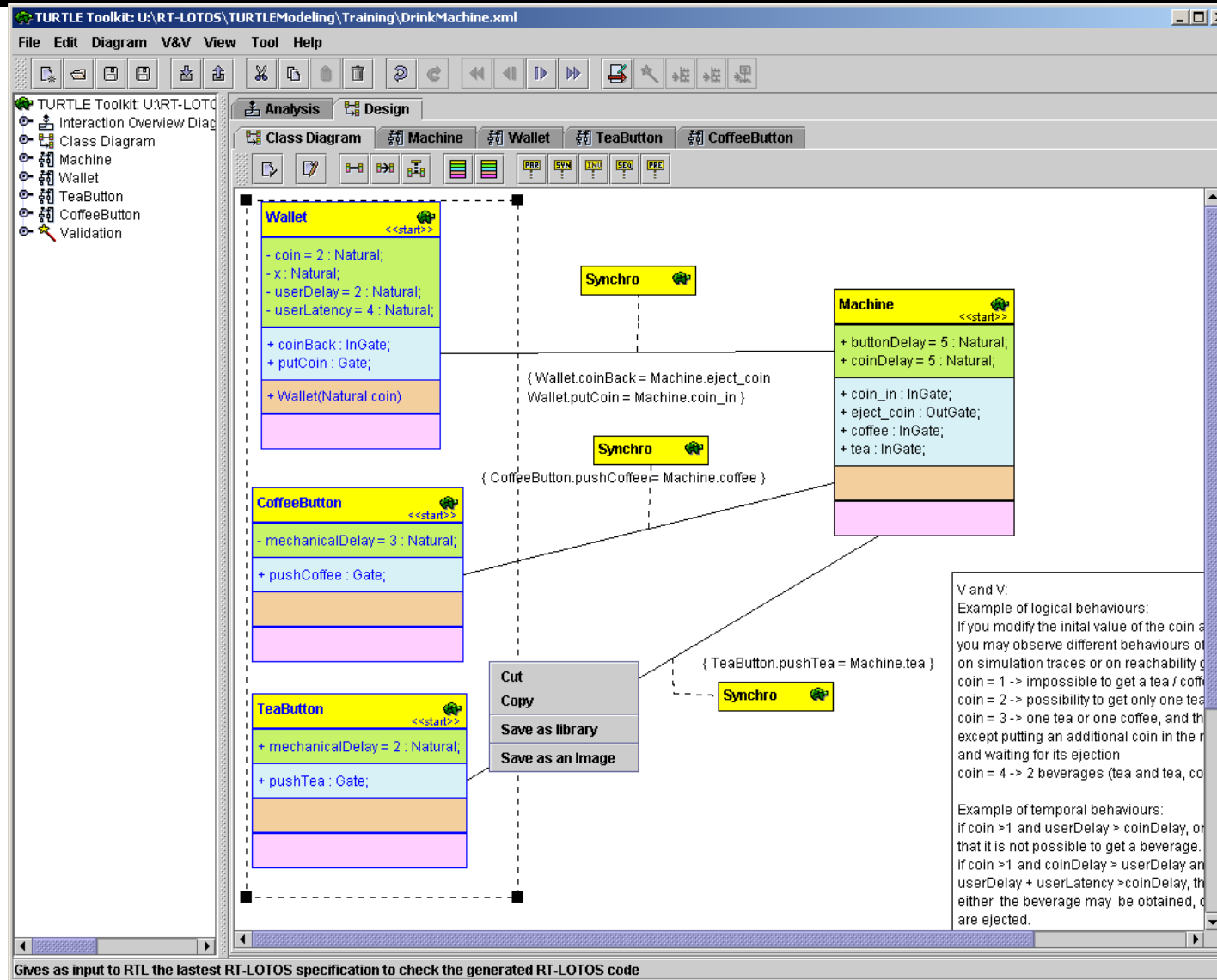
- 📄 Libraries = file in which parts of diagrams are stored
- 📄 Example: classes, activity diagrams, etc.
- 📄 From TTool:
 - ❑ *Creation of libraries*
 - Select components
 - Menu File, “Export library”
 - ❑ *Inclusion of libraries*
 - Put the mouse pointer in the diagram in which you wish to import the library and Make a right click, select “Insert library”
 - Or, menu File, “Import library”
- 📄 Only way to copy/paste diagrams from a modeling to another

Dealing with Libraries: Example



Paste
Insert Library
Increase horizontal size
Increase vertical size
Decrease horizontal size
Decrease vertical size
Rename diagram
Delete diagram

Creating a Library



I. Introduction

 **TTool: main features**

 **Installing TTool**

 **Diagramming with TTool**

  **Formal validation**

 **Java code generation**

What is Validation?

Simulation

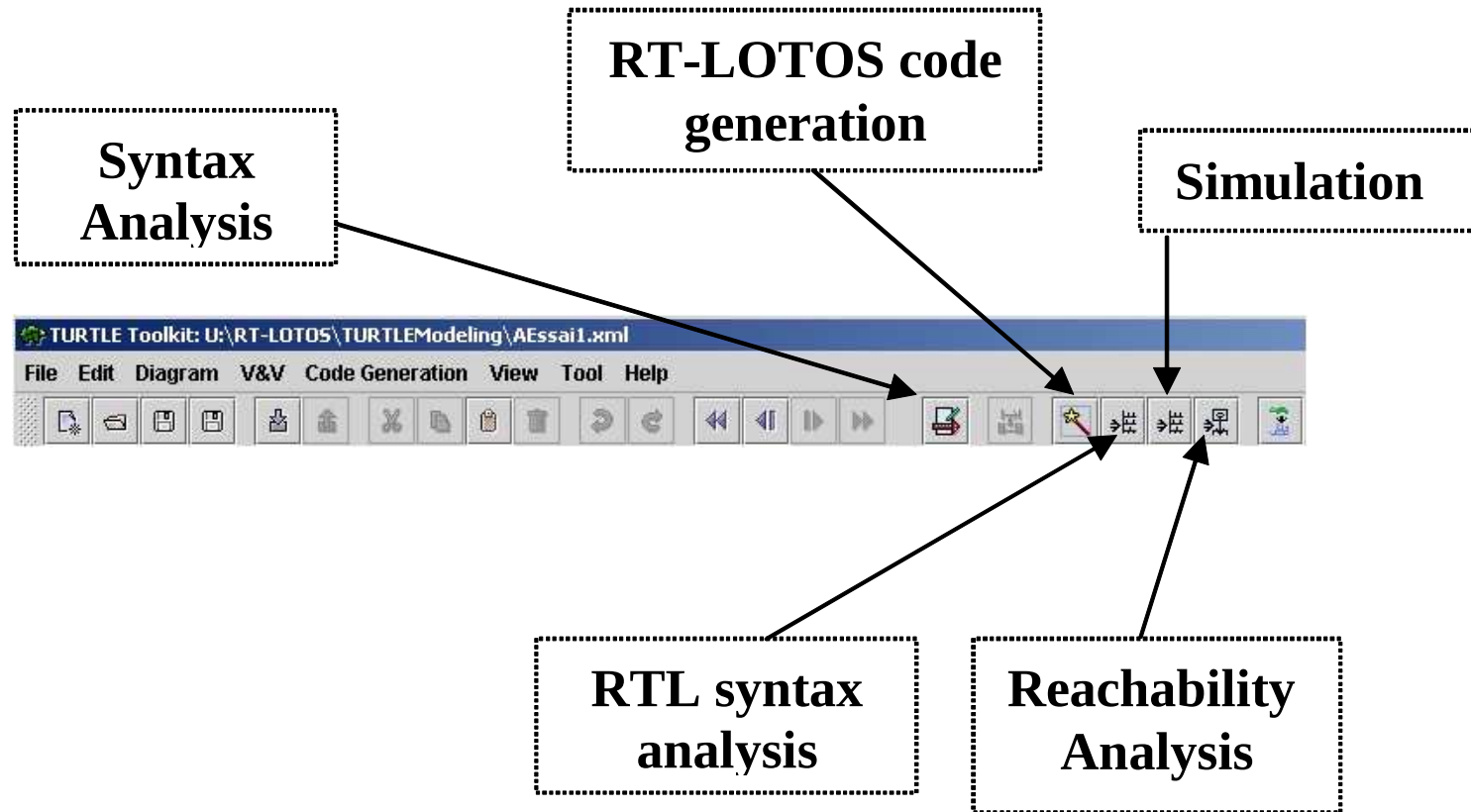
- ❑ *Partial exploration of the system*
- ❑ *Used for debugging purposes*
- ❑ *Increases confidence in a design when the system's state space cannot be explored exhaustively*

Validation / Verification

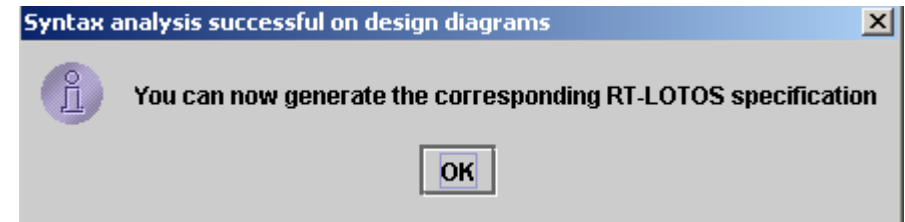
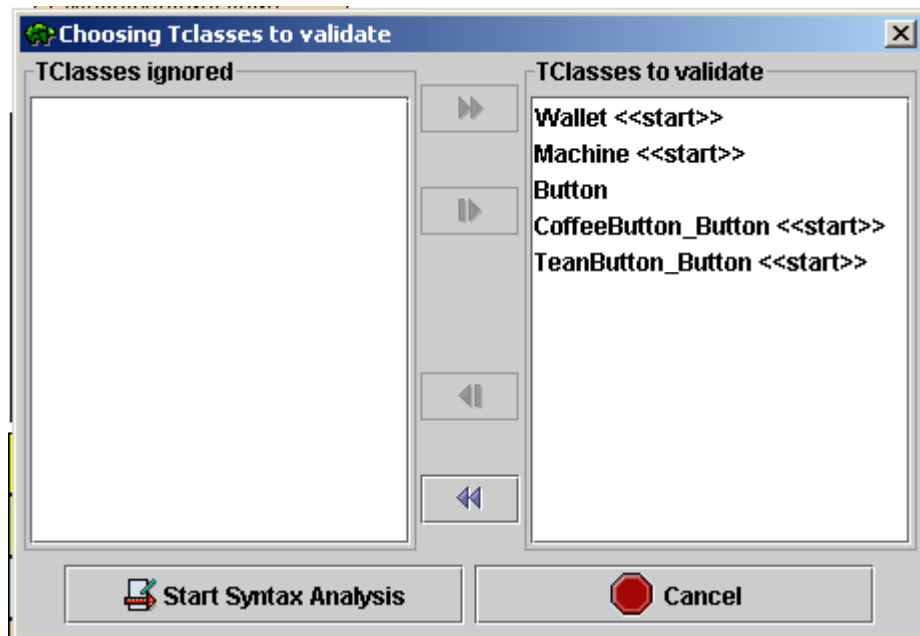
- ❑ *For bounded systems of reasonable size, exhaustive analysis becomes possible*
- ❑ *Verification relies on the whole exploration of the system's state space in order to demonstrate deadlock freeness and other properties that should be satisfied by any system*
- ❑ *Validation also relies on exhaustive analysis to demonstrate that a model meets specific requirements*



Simulation / Validation Process (Cont.)



Syntax Analysis (From a Design)



OR



Syntax Analysis (From a Design): Errors

The screenshot shows the TURTLE Toolkit interface. The left pane displays a project tree with the following items: Interaction Overview Diagram, Class Diagram, Machine, Wallet, TeaButton, CoffeeButton, Validation, Validated TClasses, Syntax analysis, and Actions --> TURTLE gates. The 'Syntax analysis' item is highlighted with a red oval, and a red arrow points to it from the text 'Errors are listed here'. The right pane shows a state machine diagram with a state transition and a 'mechanicalDelay' block. The bottom status bar reads 'Checks that all diagrams follows the TURTLE's syntax'.

Errors are listed here
(You can click on them to find which diagram is involved)

Generation of an RT-LOTOS Specification

Click on the magic stick



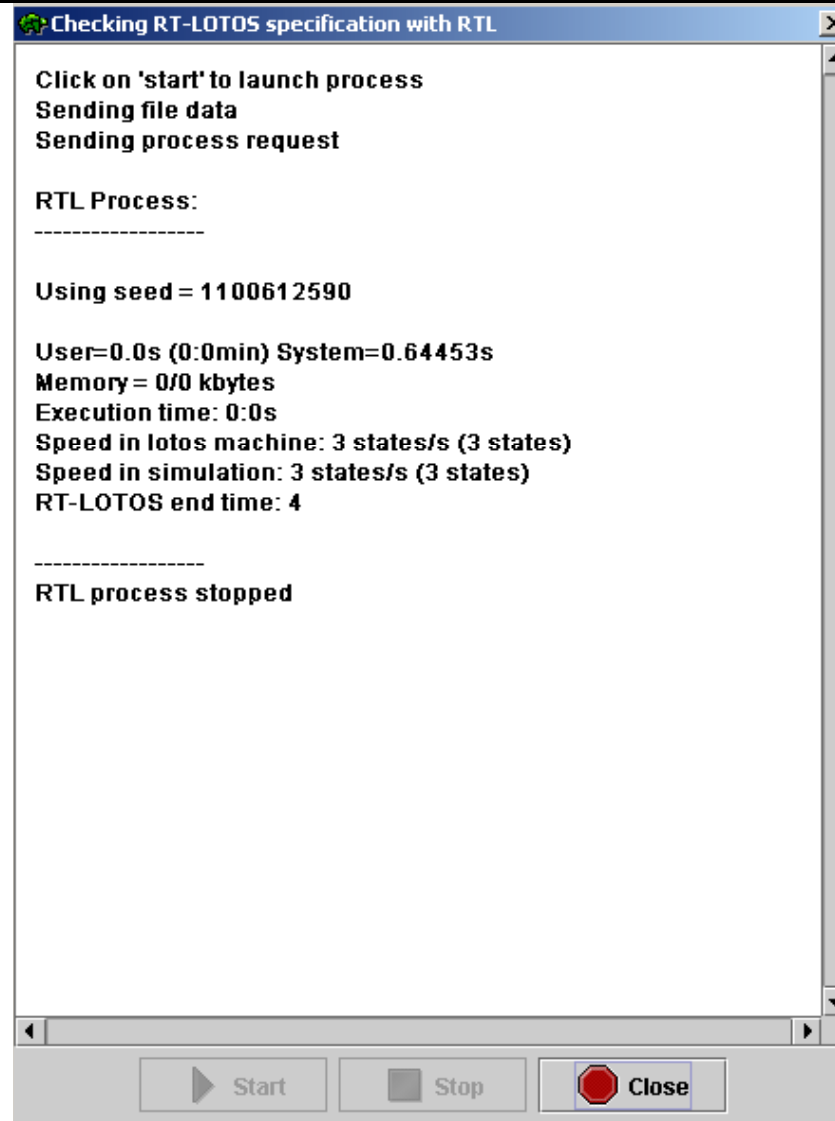
You can visualize the RT-LOTOS specification if needed, and save it from

- ❑ *Menu “View”*
- ❑ *“Show last RT-LOTOS specification”*
- ❑ *Advice: for you, this visualizing this specification is useless*

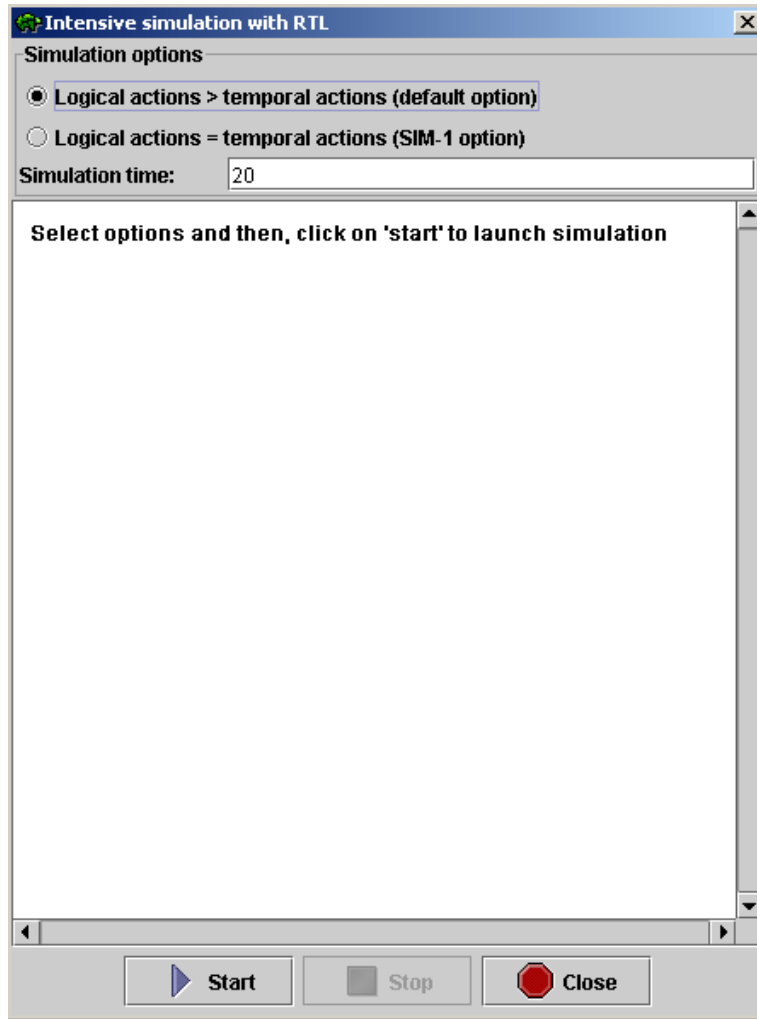
RT-LOTOS Code Checking

- ❏ **The syntax analysis of TURTLE models that you applied before generating RT-LOTOS code avoids the most common errors.**
- ❏ **Nevertheless, the current version of TTool does not implement a full syntax analysis.**
- ❏ **Before starting simulation or a reachability analysis, it is advisable to use the Check RT-LOTOS Code icon. The code is analyzed by RTL.**
- ❏ **This takes the form of a one second simulation with RTL**

RT-LOTOS Code Checking (Cont.)



Simulation



Simulation options

**Maximum number of
time units used for
simulation**

**Information about
the simulation with
RTL, and results**

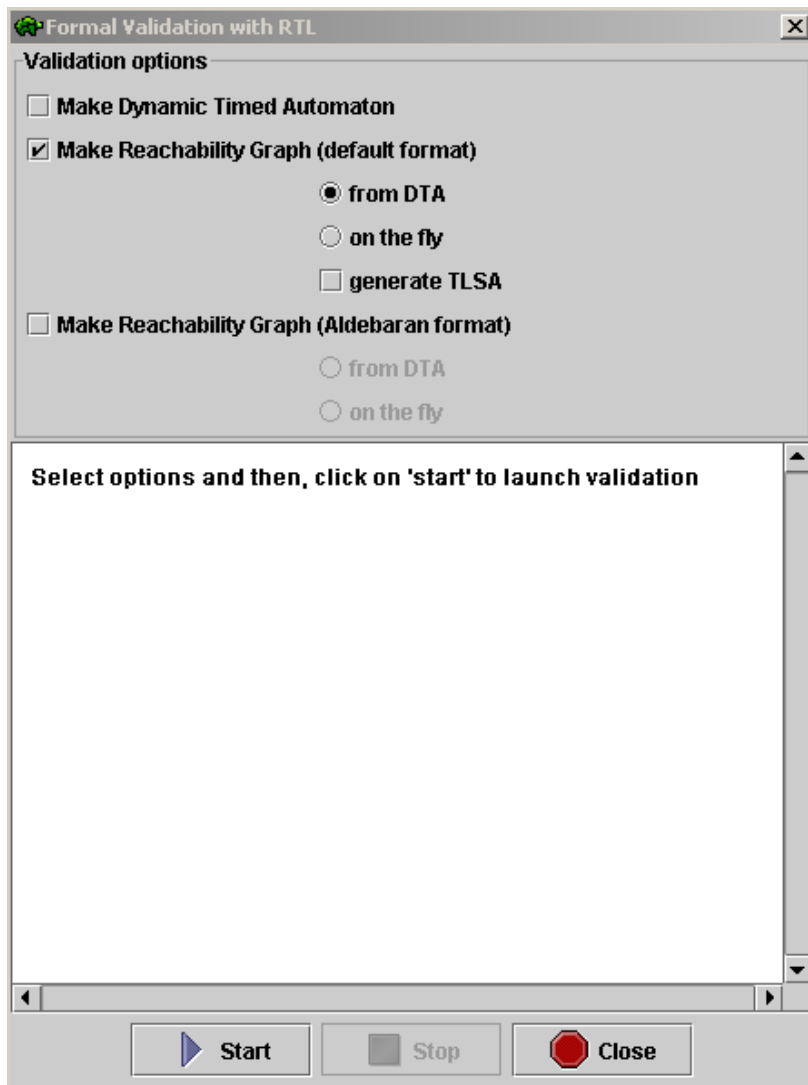
Analyzing Simulation Traces

On the left, you can see a list of paired gates

Corresponding TURTLE actions (when a click is made on a dirac)

The screenshot shows a software interface for analyzing simulation traces. At the top, a menu bar includes 'View', 'Tool', and 'Help'. The 'View' menu is open, displaying several options: 'Show last RT-LOTOS specification', 'Show last simulation trace' (highlighted with a red circle), 'Show last DTA', 'Show last RG', 'Show last TLSA', 'Show last RG (aldébaran)', 'View projected RG', 'View RT-LOTOS specification', and 'View saved graph'. Below the menu, a window titled 'Last simulation trace' is visible. It features a header bar with fields for 'Time:', 'TURTLE action(s):', 'Values:', 'RT-LOTOS action:', and 'Action No:'. The main area of the window displays a list of paired gates: 'Machine.eject_coin Wallet.coinBack' and 'Machine.coin_in Wallet.putCoin'. A red circle highlights this list. To the right of the list, a timeline with vertical red markers at 5, 10, and 15 is shown. At the bottom of the window, there are three buttons: 'Zoom -', 'Zoom +', and 'Close'.

Generating a Reachability Graph (RG)

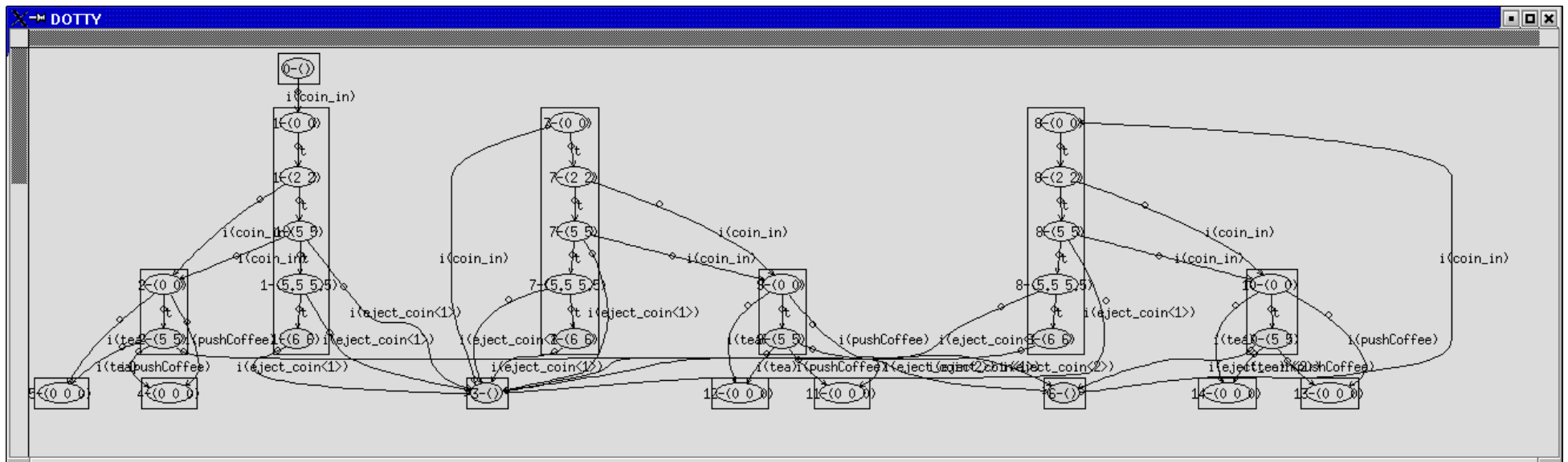
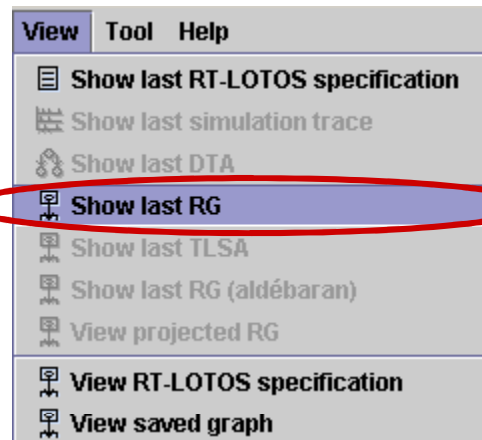
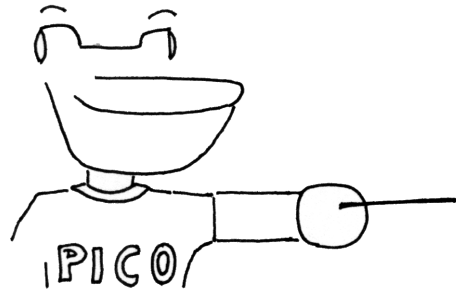


**Graph with temporal
information**

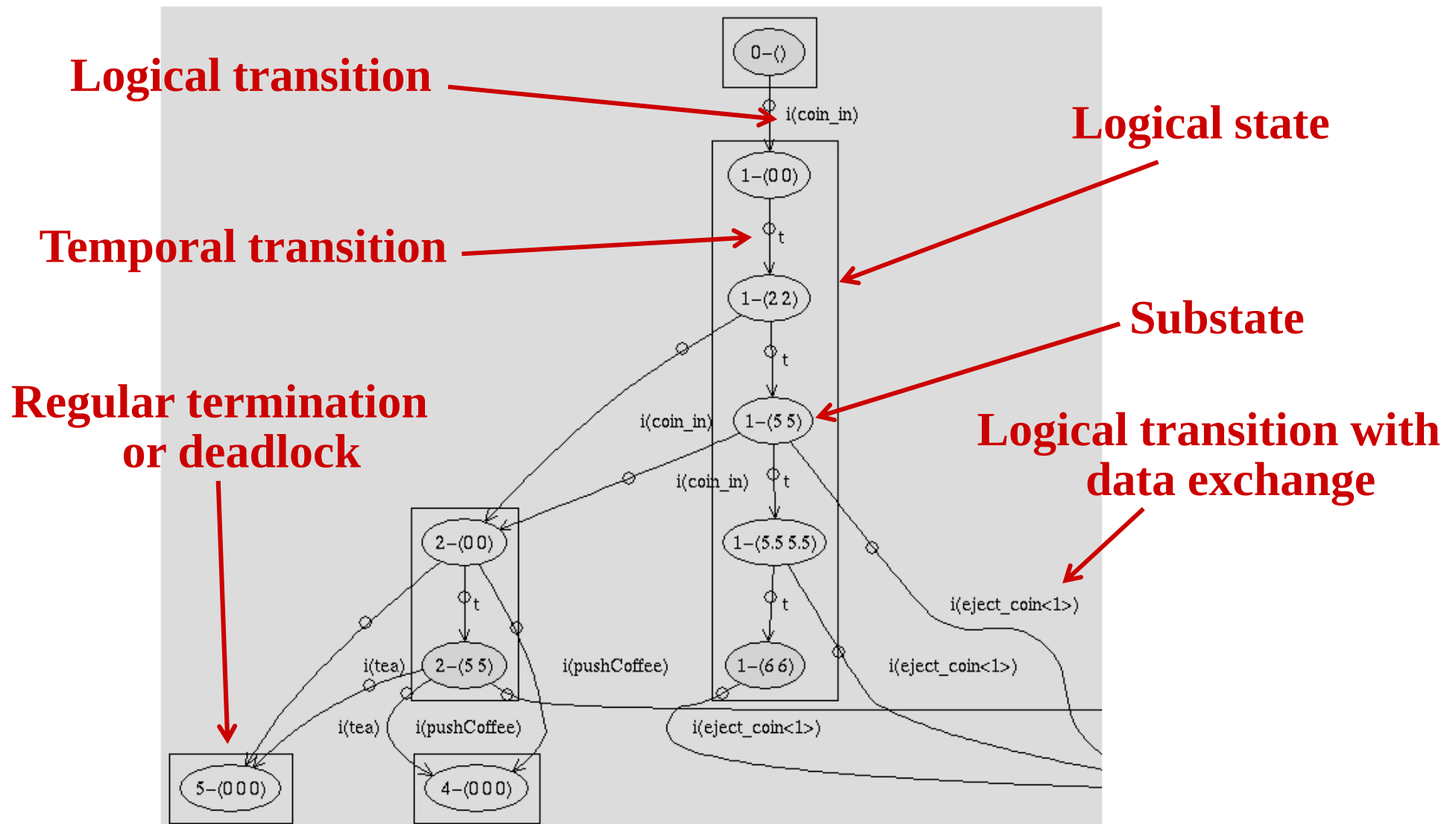
**Graph without temporal
information. Only graph
that can be used as input
for projection /
minimization**

**Information about
the generation of
RG with RTL, and
results**

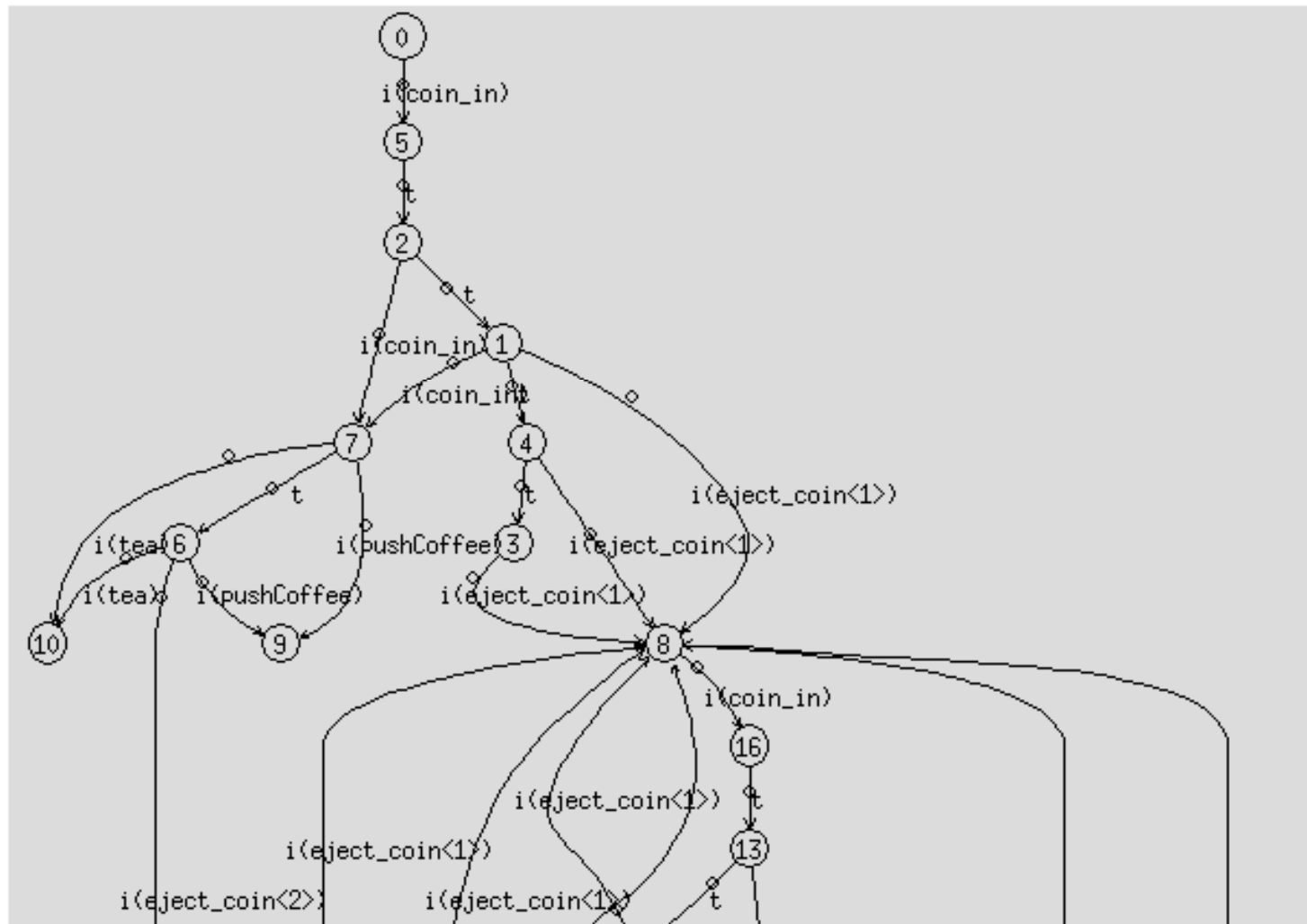
Visualization of Graphs (Default Format)



Visualization of Graphs (Cont.)



Visualization of Graphs: AUT Format



Analysis of Graphs: AUT Format

The TURTLE Toolkit interface shows the V&V menu with options: Syntax analysis (Ctrl-F5), Generate RT-LOTOS (Ctrl-F6), Check RT-LOTOS code, Run intensive simulation (Ctrl-F7), Run formal validation (Ctrl-F8), Make projection (Ctrl-F9), Make bisimulation (Ctrl-F10), Analysis (last AUT graph), Analysis (last projected AUT graph), and Analysis (saved AUT graph). A 'Synchronisation' button is also visible.

The 'Analysis on the last RG (AUT format)' dialog box shows the 'General info.' tab with the following statistics:

States	(origin, action)
10	(6, tea), (7, tea)
26	(22, pushCoffee), (23, pushCoffee)
27	(22, tea), (23, tea)
28	(24, pushCoffee), (25, pushCoffee)
29	(24, tea), (25, tea)
9	(6, pushCoffee), (7, pushCoffee)

The dialog box also includes a 'Close' button.

The 'Analysis on the last RG (AUT format)' dialog box shows the 'Statistics' tab with the following data:

Transition	Nb	(origin, action)
coin_in	9	(0, 5), (1, 7), (2, 7), (8, 16), (11, 21), (12, 23), (13, 23), (17, 23), (18, 23), (19, 23), (20, 23), (21, 23), (22, 23), (23, 23), (24, 23), (25, 23), (26, 23), (27, 23), (28, 23), (29, 23)
eject_coin<1>	9	(1, 8), (3, 8), (4, 8), (12, 8), (14, 8), (15, 8), (17, 8), (19, 8), (21, 8), (22, 8), (23, 8), (24, 8), (25, 8), (26, 8), (27, 8), (28, 8), (29, 8)
eject_coin<2>	3	(6, 11), (22, 11), (24, 11)
pushCoffee	6	(6, 9), (7, 9), (22, 26), (23, 26), (24, 28), (25, 28)
t	15	(1, 4), (2, 1), (4, 3), (5, 2), (7, 6), (12, 15), (13, 12), (15, 14), (17, 15), (18, 15), (19, 15), (20, 15), (21, 15), (22, 15), (23, 15), (24, 15), (25, 15), (26, 15), (27, 15), (28, 15), (29, 15)
tea	6	(6, 10), (7, 10), (22, 27), (23, 27), (24, 29), (25, 29)

The dialog box also includes a 'Close' button.

The 'Analysis on the last RG (AUT format)' dialog box shows the 'Shortest Paths' tab with the following information:

Shortest path from 0 to 0

Compute

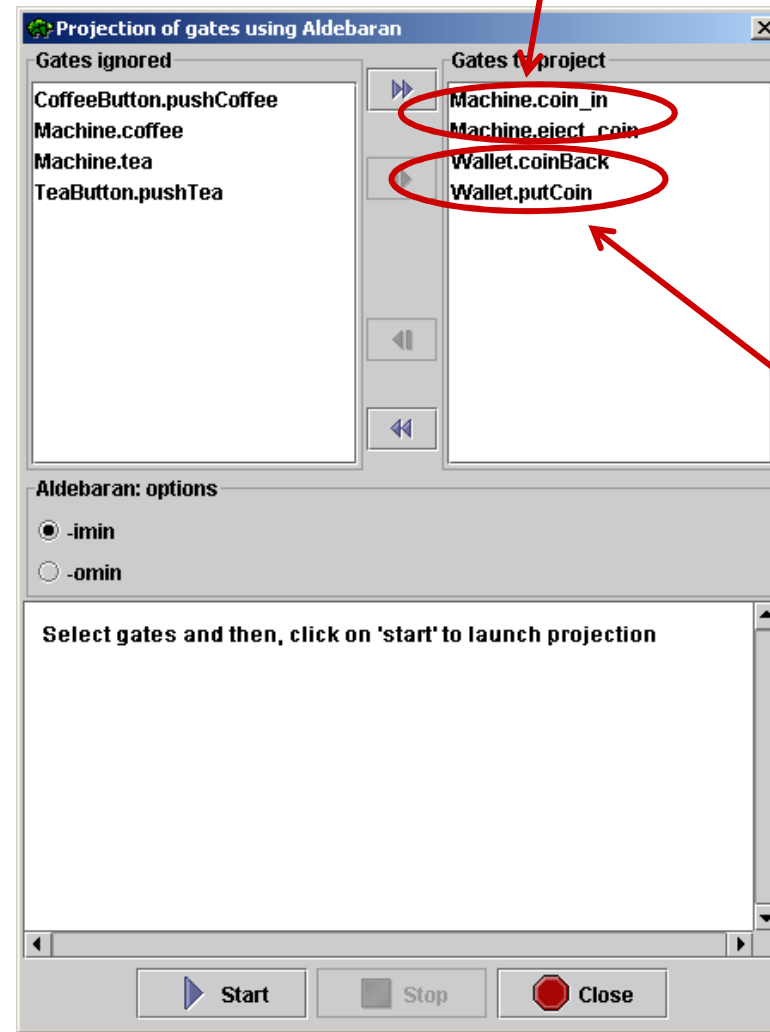
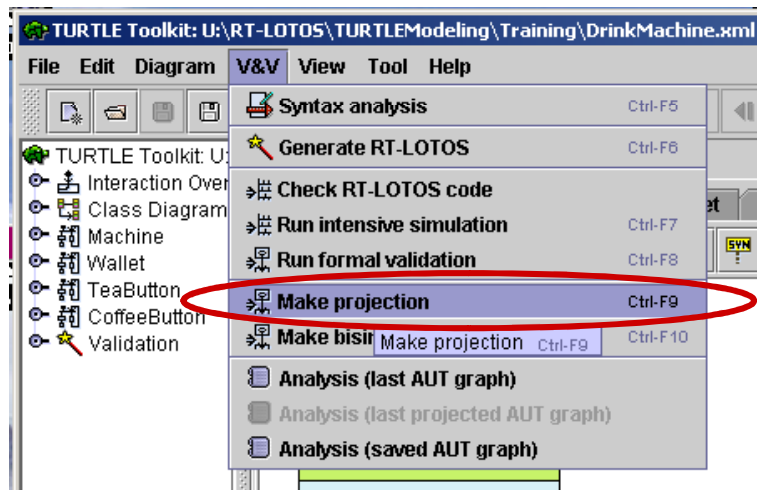
The dialog box also includes a 'Close' button.

Projections (Aldebaran)

Gates synchronized with
the selected gates

Reduce the graph to a set of
actions

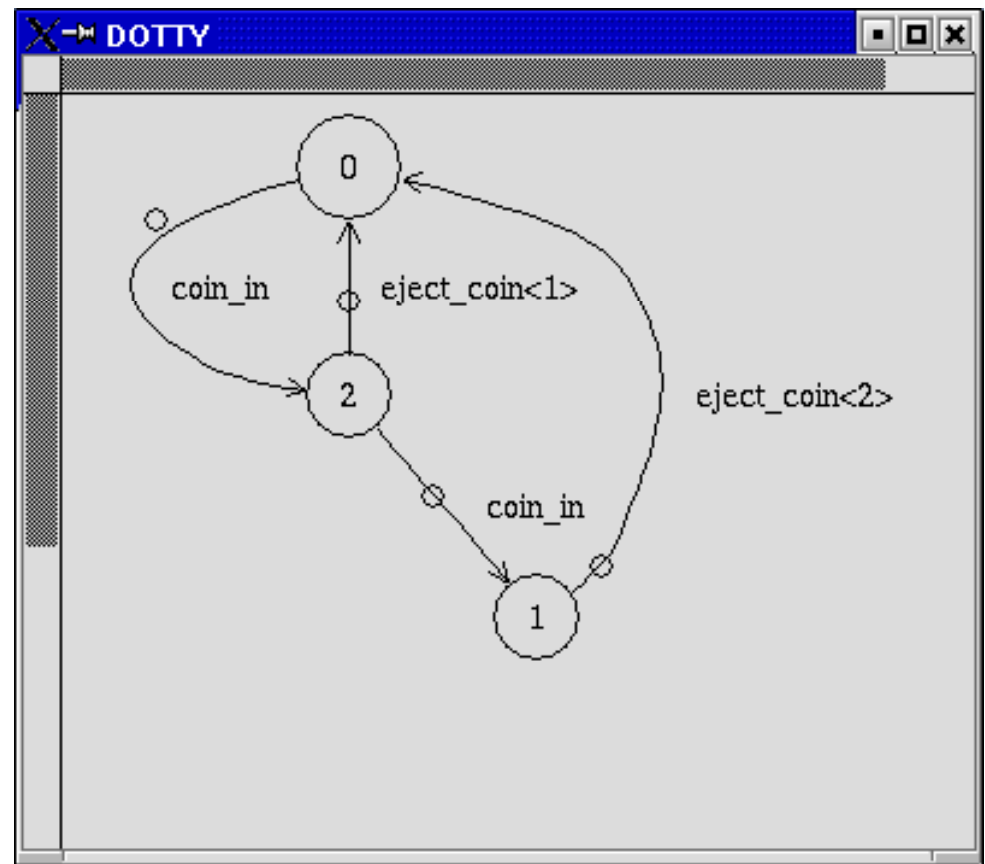
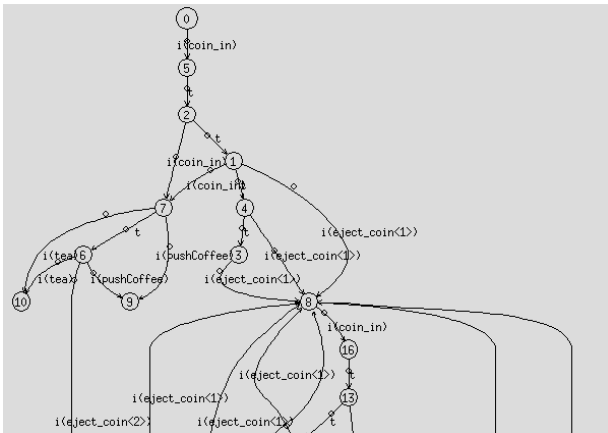
Minimize the graph



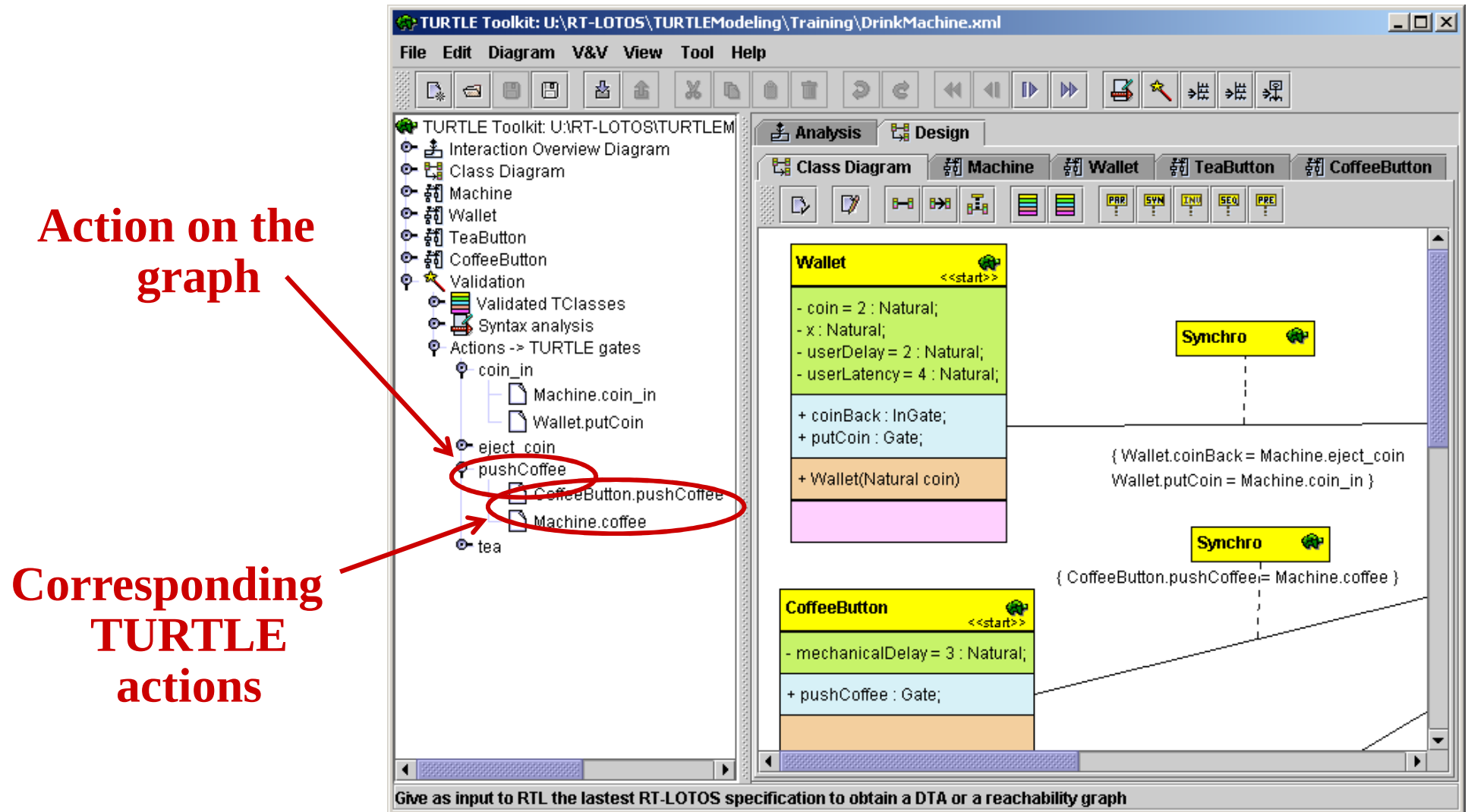
Selected
gates

Projected Graph

Projection and Minimization



From Graphs to TURTLE Actions

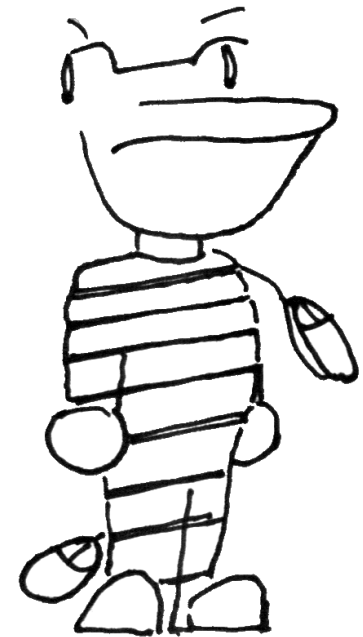


Common Modeling Errors

- ❏ Infinite loops
- ❏ Number of actions generated at $t=0$ is infinite
- ❏ -> Simulation cannot be performed

- ❏ Use of many attributes / variables
- ❏ Use of asynchronous messages (analysis)
- ❏ Use of many non-deterministic delays and choices
- ❏ -> Generation of reachability graph is difficult / long (combinatory explosion)

- ❏ Non-implementable scenarios (analysis)
 - ❑ -> Abnormal deadlock situations or nothing particular except that traces on the graph / simulation traces do not match specified traces



Observing Properties

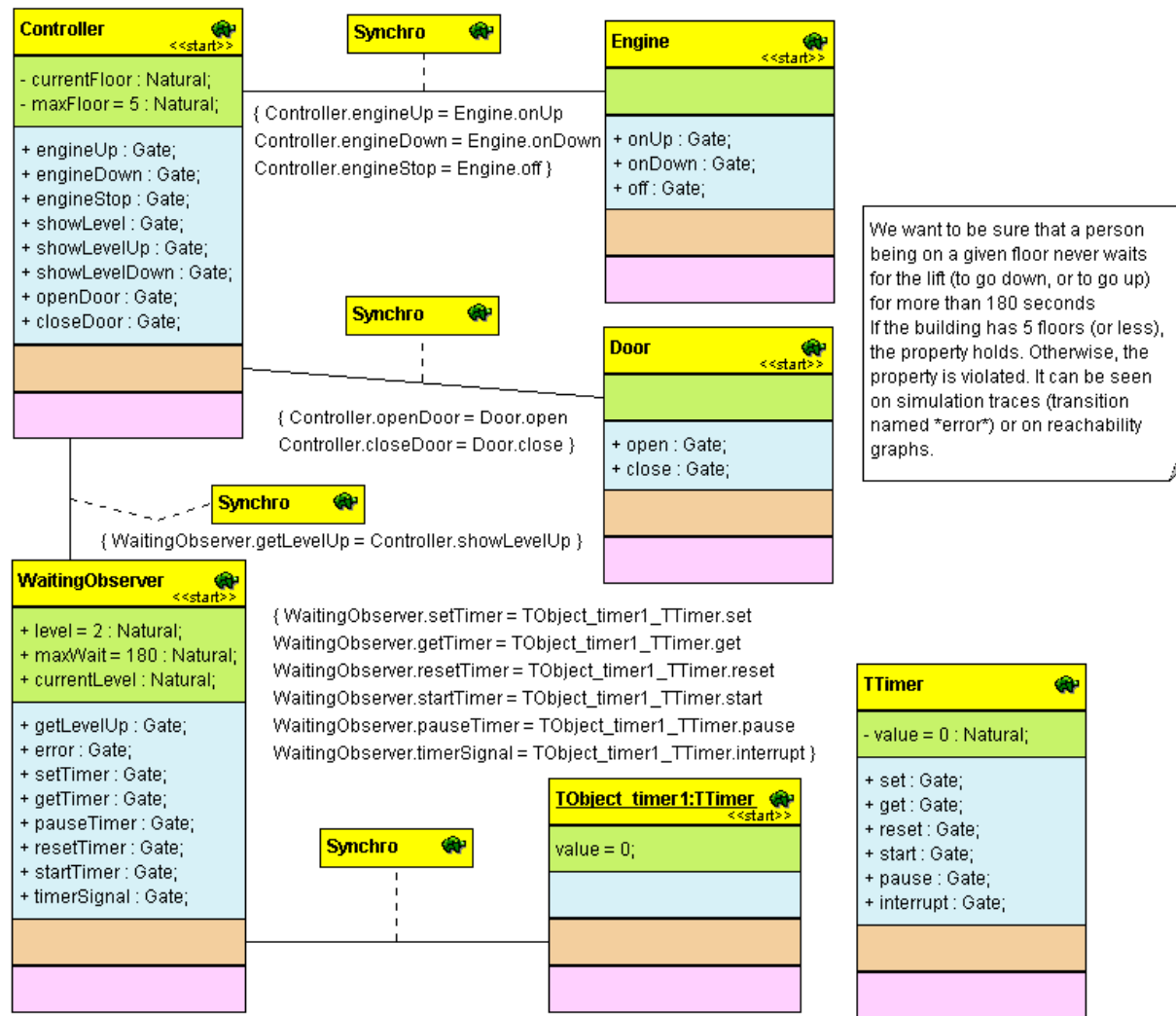
Model checking

- ❑ *kronos*

Observers

- ❑ *Avoid generating the full graph -> cut branches before they are generated*
- ❑ *Modeled as Tclasses at class diagram level*
- ❑ *Should be non intrusive*
- ❑ *Should not induce an important overhead when generating the graph*
 - Keep them simple!
- ❑ *Should stop the application under verification when a property violation is detected*
 - “error” transition
 - Observer should synchronize with other classes to stop them

Example of Property Observation



I. Introduction

 **TTool: main features**

 **Installing TTool**

 **Diagramming with TTool**

 **Formal validation**

  **Java code generation**

Fundamentals of Java Code Generation

Java code may be generated from

TURTLE designs

- Monolithic application -> 1 executable application

TURTLE deployments

- Distributed application -> 1 executable application / execution node

Steps

Configuration of TTool

Adding of directives

Generation of code

Compilation of code

Execution of code

Configuration of TTool

config.xml

- ❑ *<JavaCodeDirectory data="U:\RT-LOTOS\JavaCode\" />*
 - Directory in which Java code is generated
- ❑ *<JavaCompilerPath data="[C:\Program Files\j2sdk_nb\j2sdk1.4.2\bin\javac.exe" />*
 - Compiler used to compile generated Java code
- ❑ *<TToolClassPath data="U:\RT-LOTOS\JavaCode" />*
 - Classpath used at compilation / execution
- ❑ *<JavaExecutePath data="[C:\Program Files\j2sdk_nb\j2sdk1.4.2\bin\java.exe" />*
 - Link to the Java virtual machine used for executing generated Java code

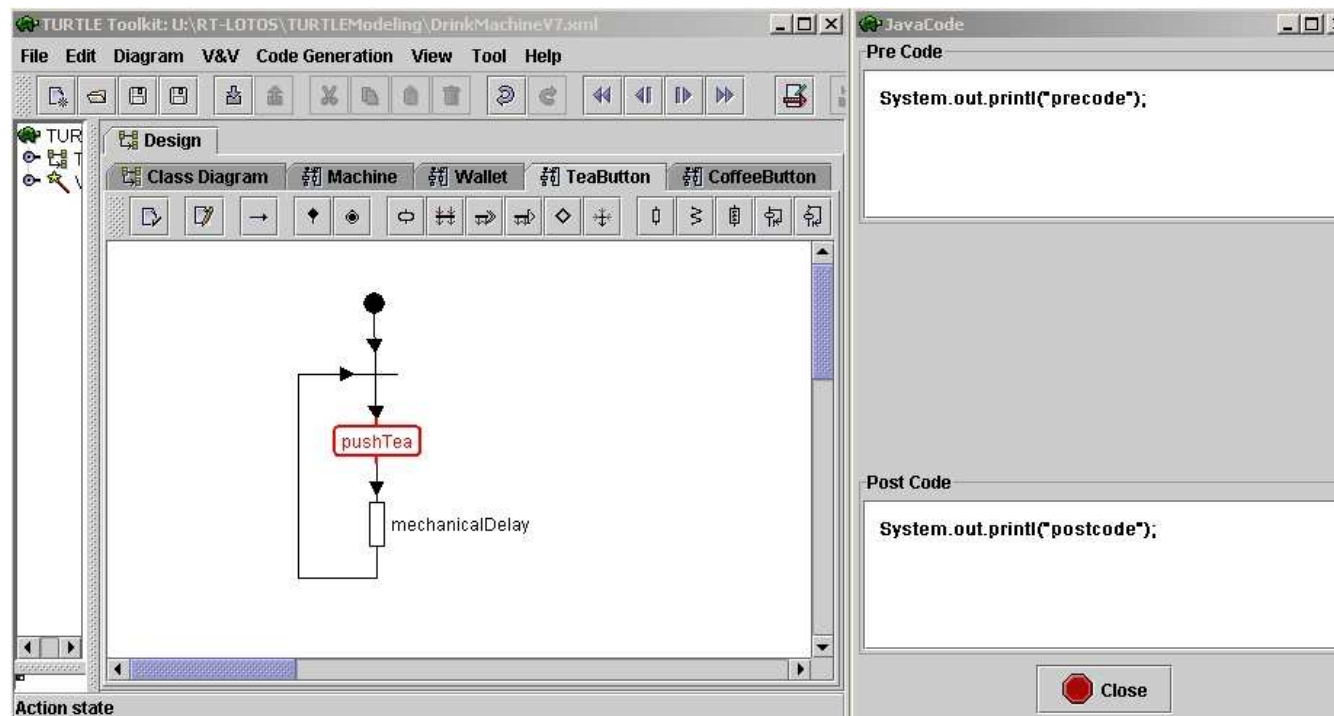
Libraries

- ❑ *Used by the generated Java code*
- ❑ *Should be located in the TToolClassPath*

Adding directives

TURTLE design

- ❑ *Code may be added on the following operators of activity diagrams*
 - Actions
 - Variable settings
- ❑ *Java code executed before the action*
- ❑ *Java code executed after the action*



Adding directives (Cont.)

TURTLE deployment

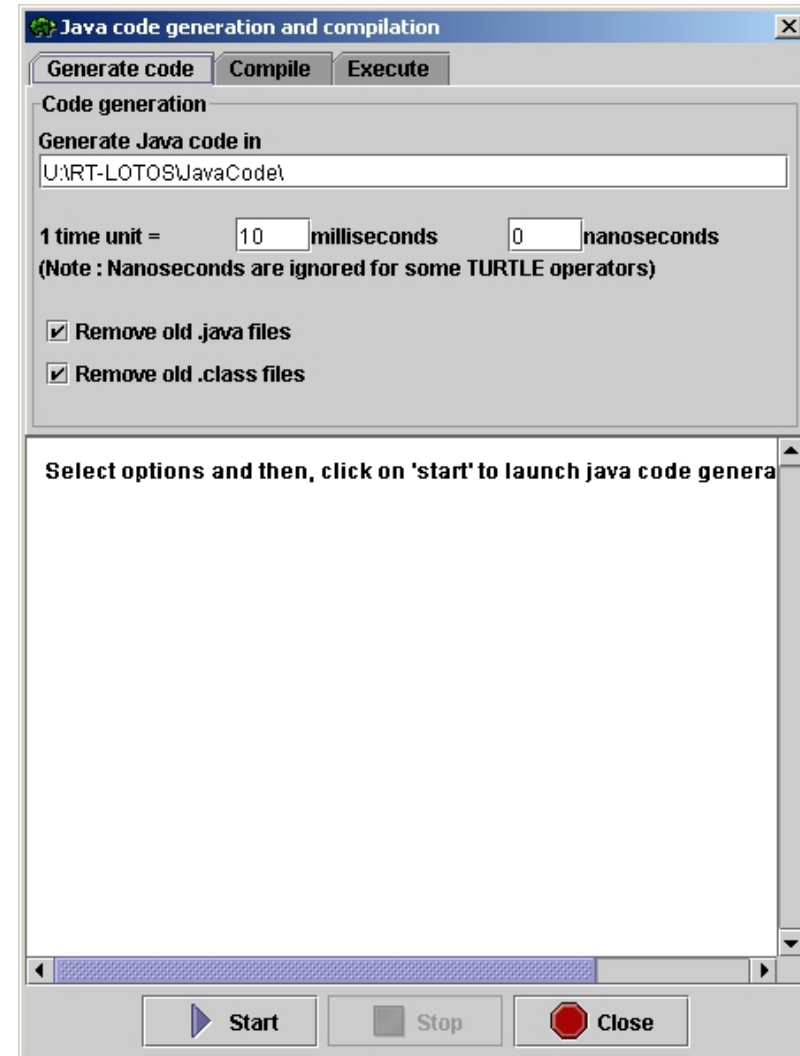
- ❑ *Code may be added to TURTLE components (considered as TURTLE designs)*
- ❑ *Communication protocols between TURTLE components may be specified on links between TURTLE nodes*
 - UDP, TCP, RMI

Info. on protocols

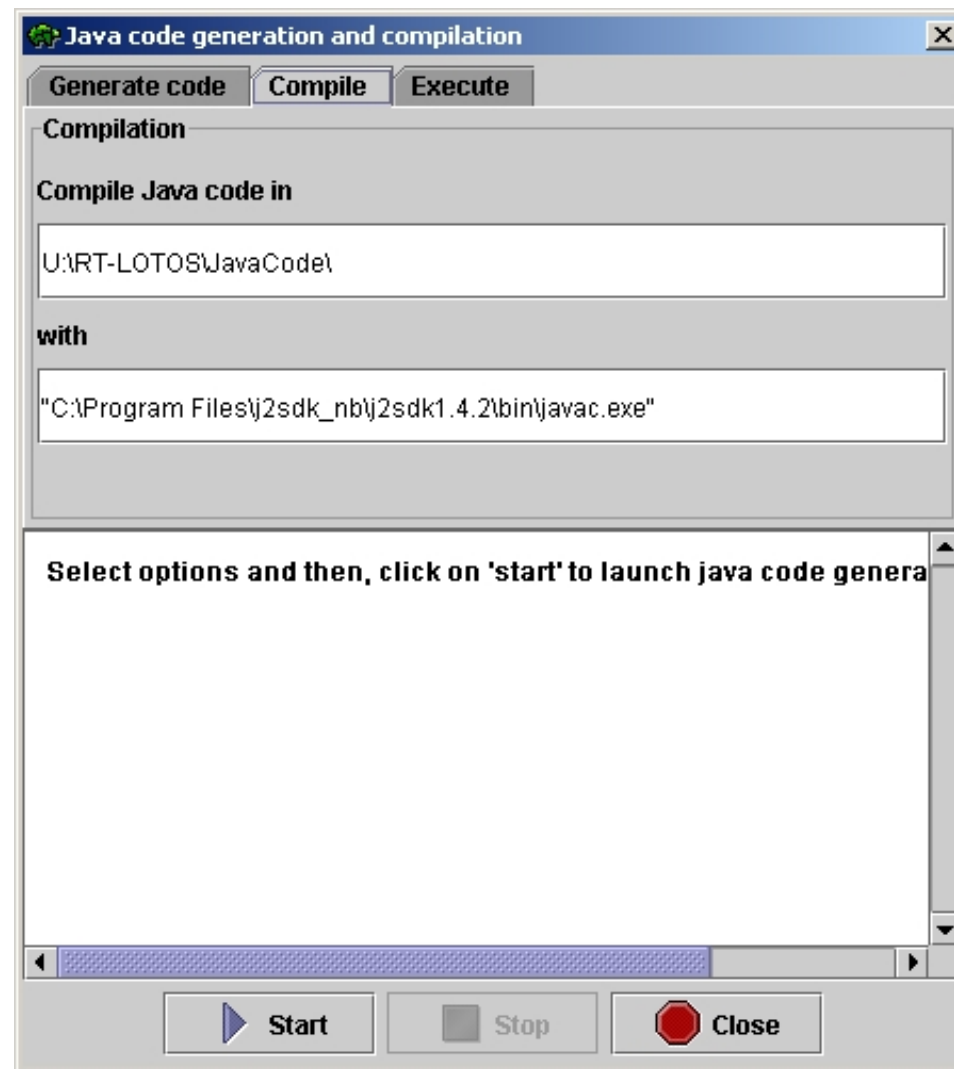


Generating Java Code

- ❏ Check the syntax of your modeling
- ❏ Then, select the *Generate JAVA* icon
- ❏ Set various options, including
 - ❑ *TURTLE time unit vs. Java time unit*
 - ❑ *Debug information*



Compiling Java code



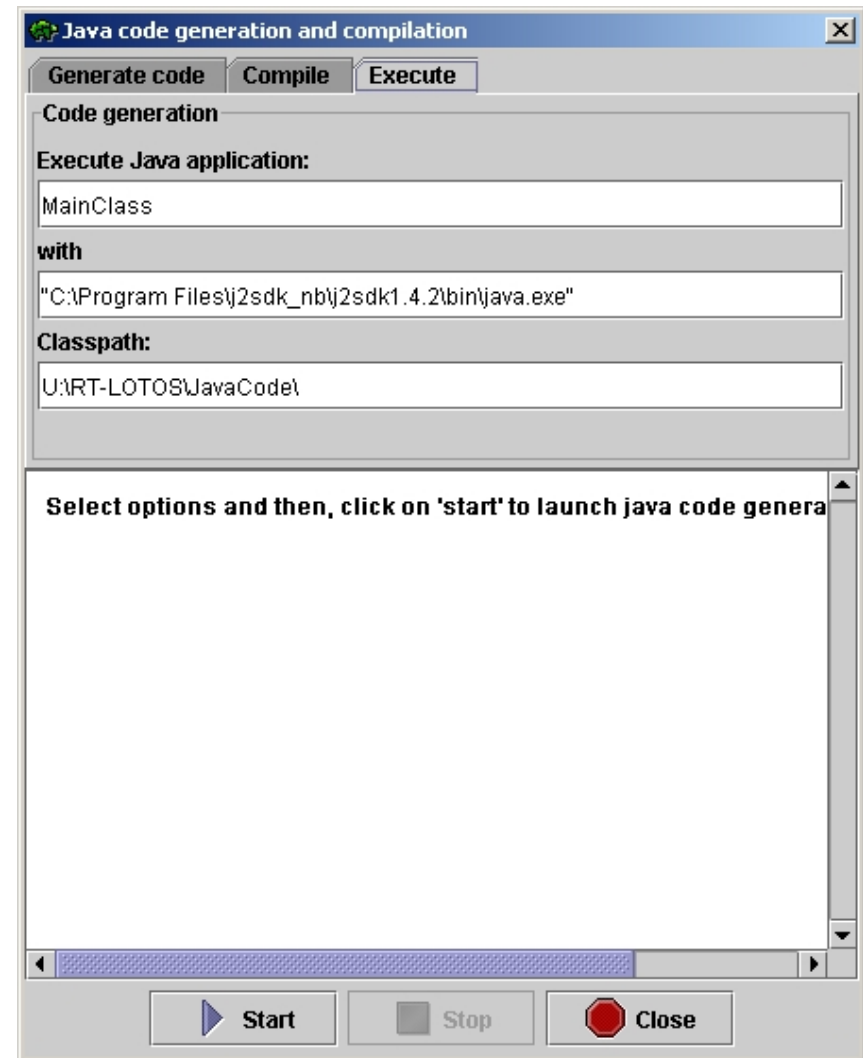
Executing Generated Java Applications

Monolithic applications

- ❑ *Generated from TURTLE designs*
- ❑ *Use of the provided graphical interface*

Distributed systems

- ❑ *Command line*
- ❑ *Various applications should be started in the right order*
 - Server side before client side
- ❑ *RMI*
 - Start *rmiregistry* first
 - Set the security policy
 - Run various instances
 - See the online help for more information



How to Improve Generated Code?

- ❏ **Do not use non deterministic choices**
 - ❑ *They are not deterministic*
 - ❑ *The Java code generator may not interpret them as you expected*
- ❏ **Creation of parallel subactivities, sequence operators, preemption operators are translated using Java threads for each subactivities**
 - ❑ *Performance drawback*

Examples / Exercises

📄 <http://www.eurecom.fr/~apvrille/TURTLE/HELP/index.html>

□ *“Example” section*

📄 **Technical support, maintenance, bug report**

□ *ludovic.apvrille@enst.fr*

