Play-In/Play-Out &
LSC to State Chart Transformation

Riandi Wiguna
rian.wiguna@mail.mcgill.ca
School of Computer Science
McGill University
August 27, 2004

"You better hit bull's eye, the kid don't play."
-Vanilla Ice, "Ice, Ice, Baby"

Overview

1. Intro to Play-In/Play-Out
2. LSCs (Live Sequence Charts)
3. Play-In
4. Play-Out
5. Example: Simple Microwave
6. LSC to Statechart Transformation

Intro to Play-In/Play-Out

The Play-In/Play-Out Approach is a way to easily generate and test LSCs (Live Sequence Charts). LSCs model all desired system reactions, providing a complete design for the system.

The basic idea is to feed both input and desired output into a “Play-Engine” which generates LSCs automatically. We then run the system through the Play-Engine, making sure the system satisfies our requirements.

Intro to Play-In/Play-Out

A step-by-step view of the Approach:

1. Determine system requirements
2. Build system GUI (graphical user interface)
3. Play-In scenarios into GUI / Play-Engine makes LSCs
4. Play-Out system through GUI, testing it / Play-Engine displays system's fidelity to LSCs throughout run

Both designers and end-users can participate in the software design process through Play-In/Play-Out.

Intro to Play-In/Play-Out

This presentation is based off "Specifying and Executing Behavioral Requirements: The Play-In/Play-Out Approach" by David Harel and Rami Marelly

and "Synthesizing State-Based Object Systems from LSC Specifications" by David Harel and Hillel Kugler.

Additional information from:

1. "DCharts, a Formalism for Modeling and Simulation Based Design for Reactive Software Systems" by Thomas Huining Feng

2. "Can Behavioral Requirements be Executed? (And why would we want to do so?)" by David Harel

(from pgs. 4-6 of “Specifying and Executing...”)

LSCs (Live Sequence Charts)

- Modified MSCs (Message Sequence Charts)
- LSCs model system reactions that *must* happen as well as those that just *may* happen
- LSCs model messages that *must* be sent as well as those that just *may* be sent
- Two different kinds of LSCs:
  - Universal
  - Existential

Universal LSCs

- Model system reactions that *must* happen
- Drawn with solid border
- Pre-Chart is condition for main chart actions
- Violating these or exiting prematurely causes a system error/crash
- Drive system execution during Play-Out

Existential LSCs

- Model system reactions that *may* happen
- Drawn with dashed border
- Must be able to run to completion in at least one system scenario
- Monitored during Play-Out

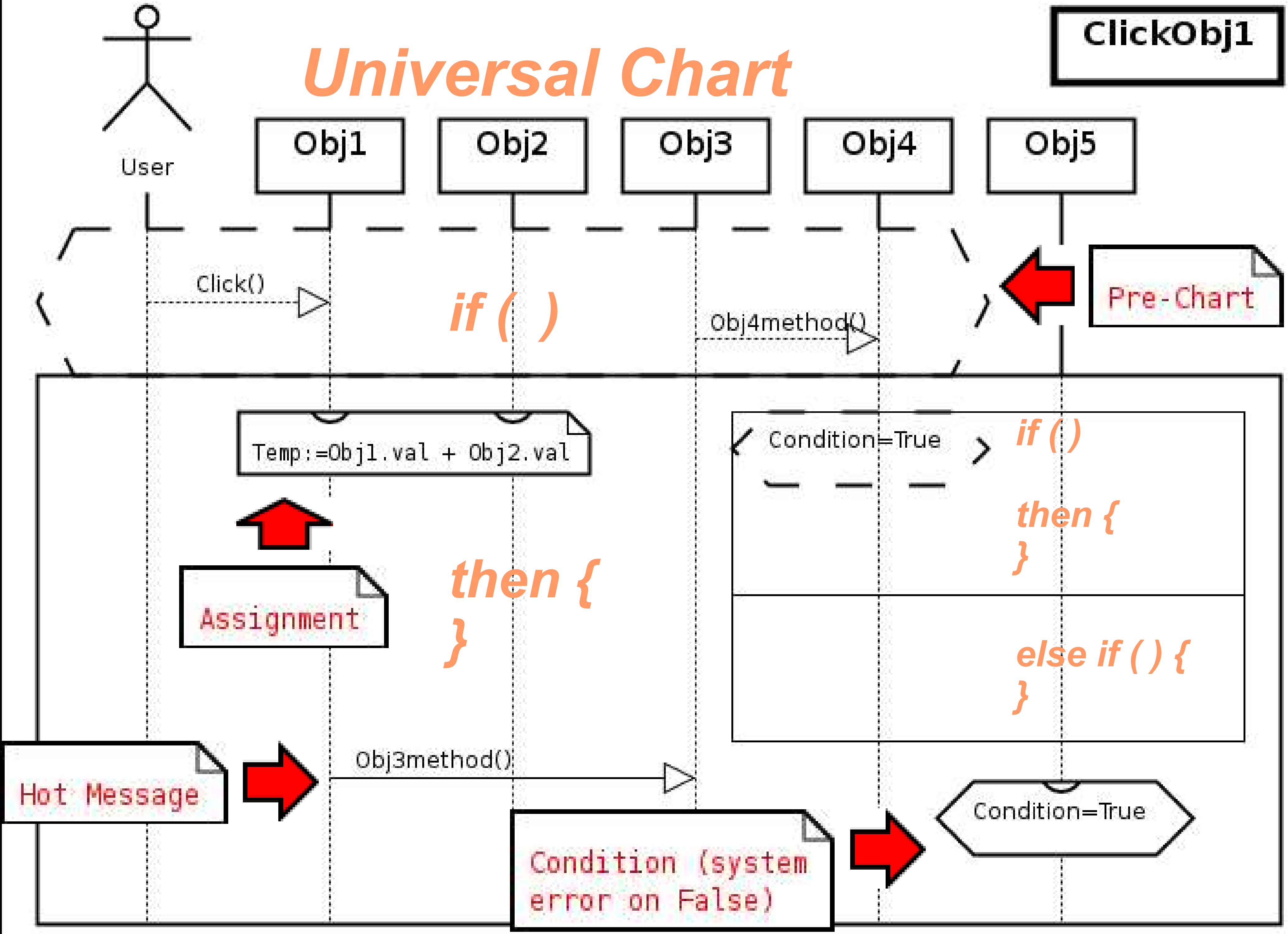
LSC Logic Symbols

- Message (Arrow)
  - Hot (solid tail, must always be sent)
  - Cold (dashed tail, may be sent)
- Condition (Hexagon)
  - Half-circles denote object synchronicity
- Loop (Rectangle)
  - Integers in corner denote predetermined number of iterations

LSC Logic Symbols

- If-Else (Dashed Hexagon in Rectangles)
  - Rectangles contain consequences of each possible outcome
- Local Variable Assignment ("Note" Rectangles)
  - Half-circles denote object dependency

Universal Chart



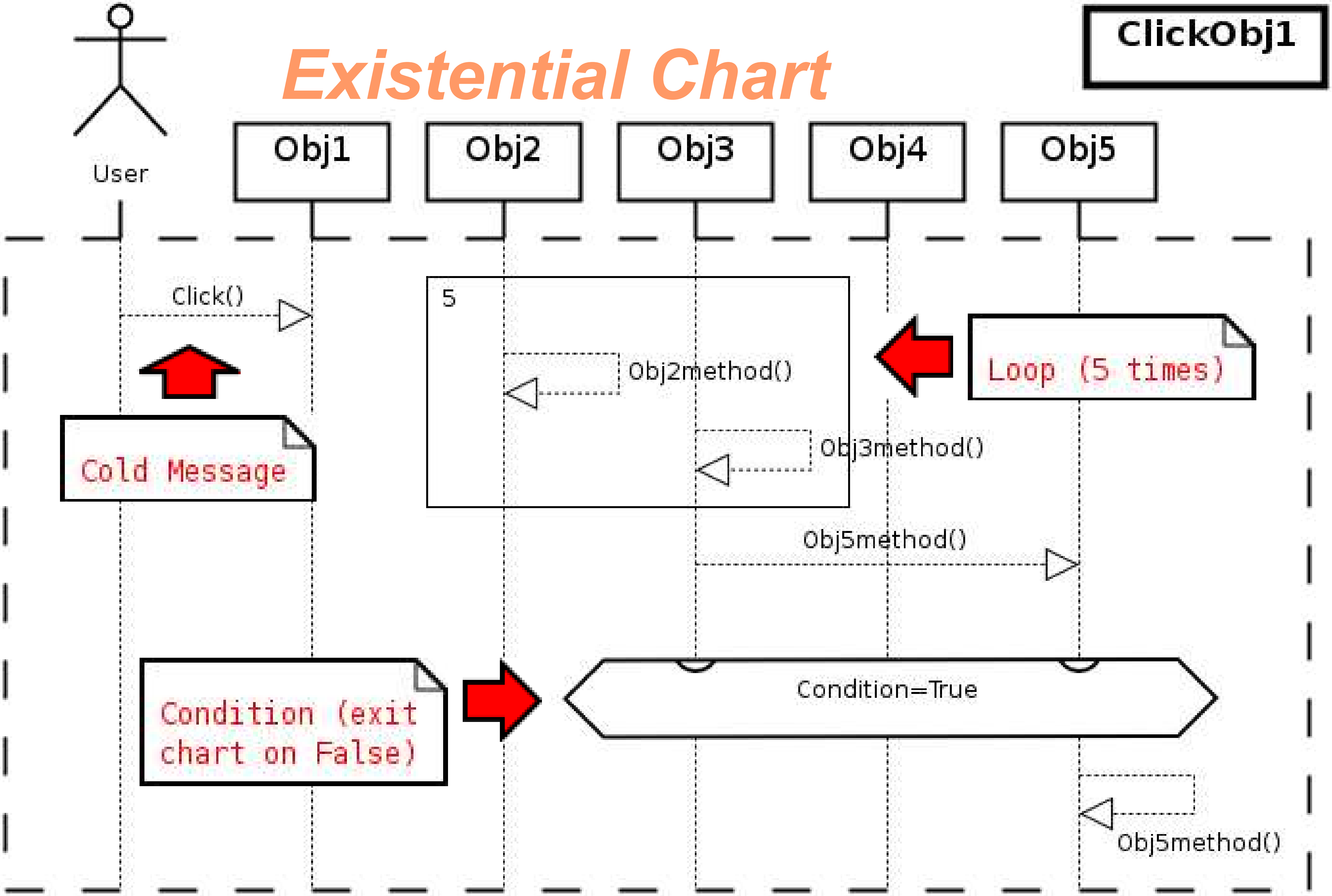
Pre-Chart

Assignment

Hot Message

Condition (system error on False)

Existential Chart



Cold Message

Loop (5 times)

Condition (exit chart on False)

(from pgs. 7-8 of “Specifying and Executing...”)

Play-In

- User only deals with GUI, not LSCs themselves
- Basic procedure:
  1. User creates use case and describes it
  2. User interacts with a GUI element as if actually running system (click buttons, highlight text, type text, etc.)
  3. User utilizes right-clicks/context menus on GUI elements to describe how they should be affected by previous interaction
  4. Play-Engine updates GUI interface and LSCs automatically
  5. User repeats steps 1-4 until all LSCs generated

Play-In

- Play-Engine provides dialogs to input information about if-else blocks, type of messages (hot or cold), and other logic symbols
- User can create functions to generalize actions (system responses to clicking digits 1-9 on a calculator)
- User can right-click a GUI element, choose “External Change” to mimic environmental inputs/effects on objects' states

(from pgs. 8-10 of “Specifying and Executing...”)

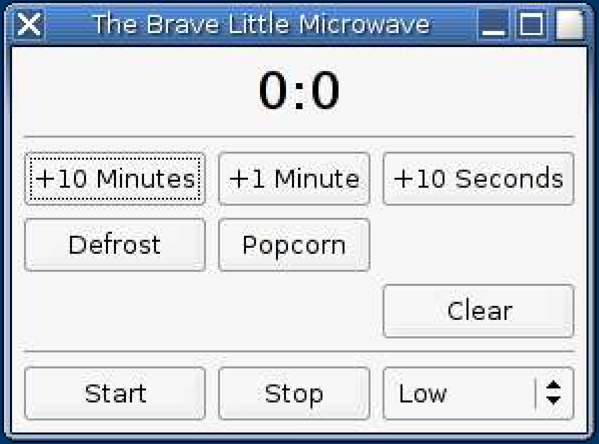
Play-Out

- The system runs as if it was fully implemented
- Displays active and monitored LSCs. User may ignore LSCs and focus on GUI
- Modes
  - Step (System stops after every reaction, waits for user input/acknowledgment)
  - Super-Step (System continues making reactions until no new ones can be made, waits for user input)

Play-Out

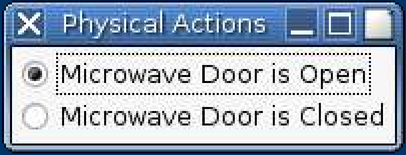
- “Cuts” show position of system in the LSCs
  - Hot (“Combed” red line, system aborts if LSC violated)
  - Cold (“Combed” blue line, system exits LSC if LSC violated)
- Play-Out runs can be saved in XML format
- LSCs are framed in blue when completed
- LSCs are crossed out when violated

Example: Simple Microwave



0:0

Low

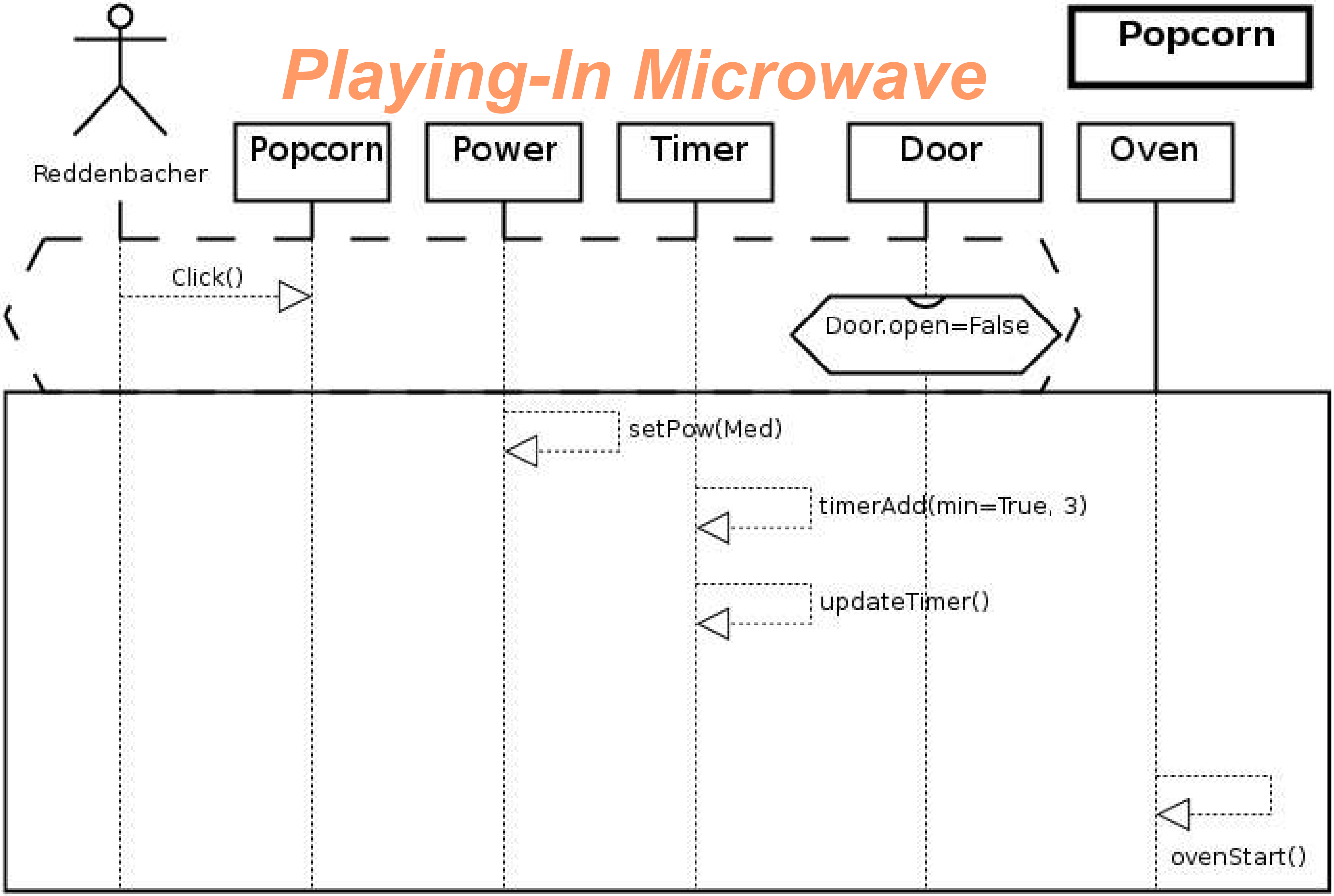


● Microwave Door is Open

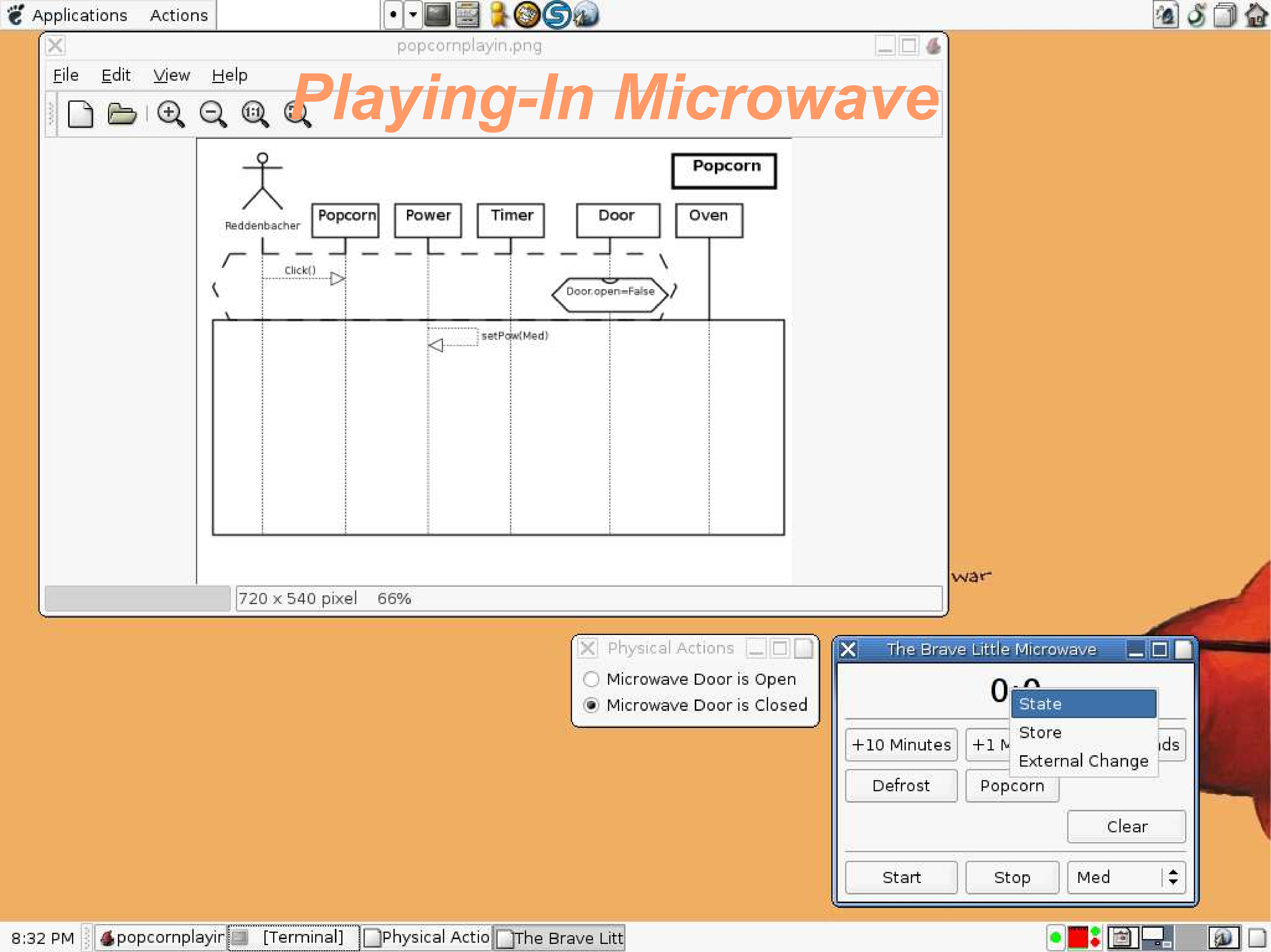
○ Microwave Door is Closed

- System Constraints:
  - Take button presses as “**TimeRemaining**”
  - Start microwave on event “**Start**”
  - Stop microwave on any of below
    - event “**Stop**”
    - event “**OpenDoor**”
    - “**TimeRemaining == 0**”
    - event “**SmokeDetected**”

Playing-In Microwave



Playing-In Microwave



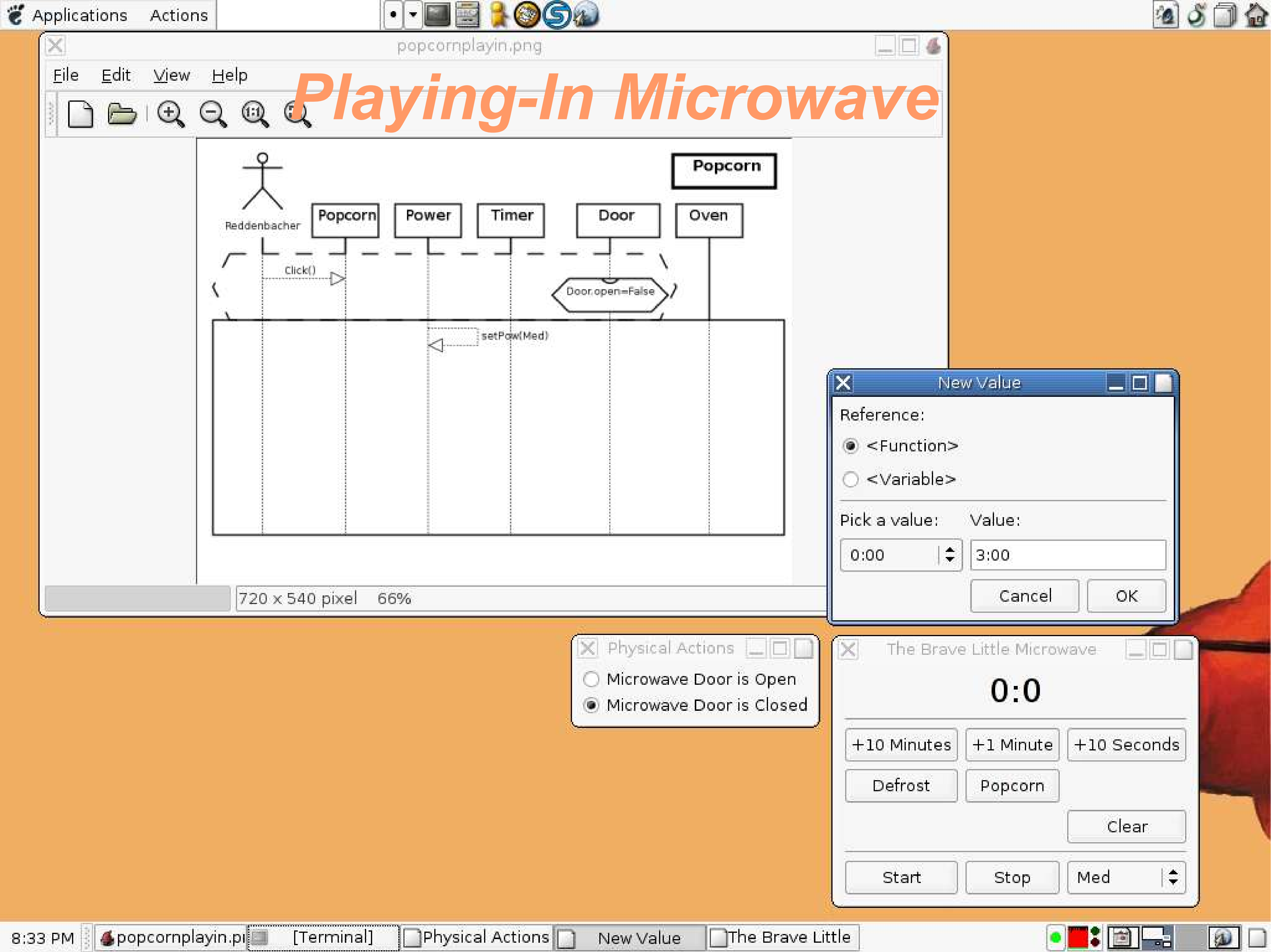
720 x 540 pixel 66%

Physical Actions

○ Microwave Door is Open

● Microwave Door is Closed

Playing-In Microwave



New Value

Reference:

● <Function>

○ <Variable>

Pick a value: 0:00

Value: 3:00

Physical Actions

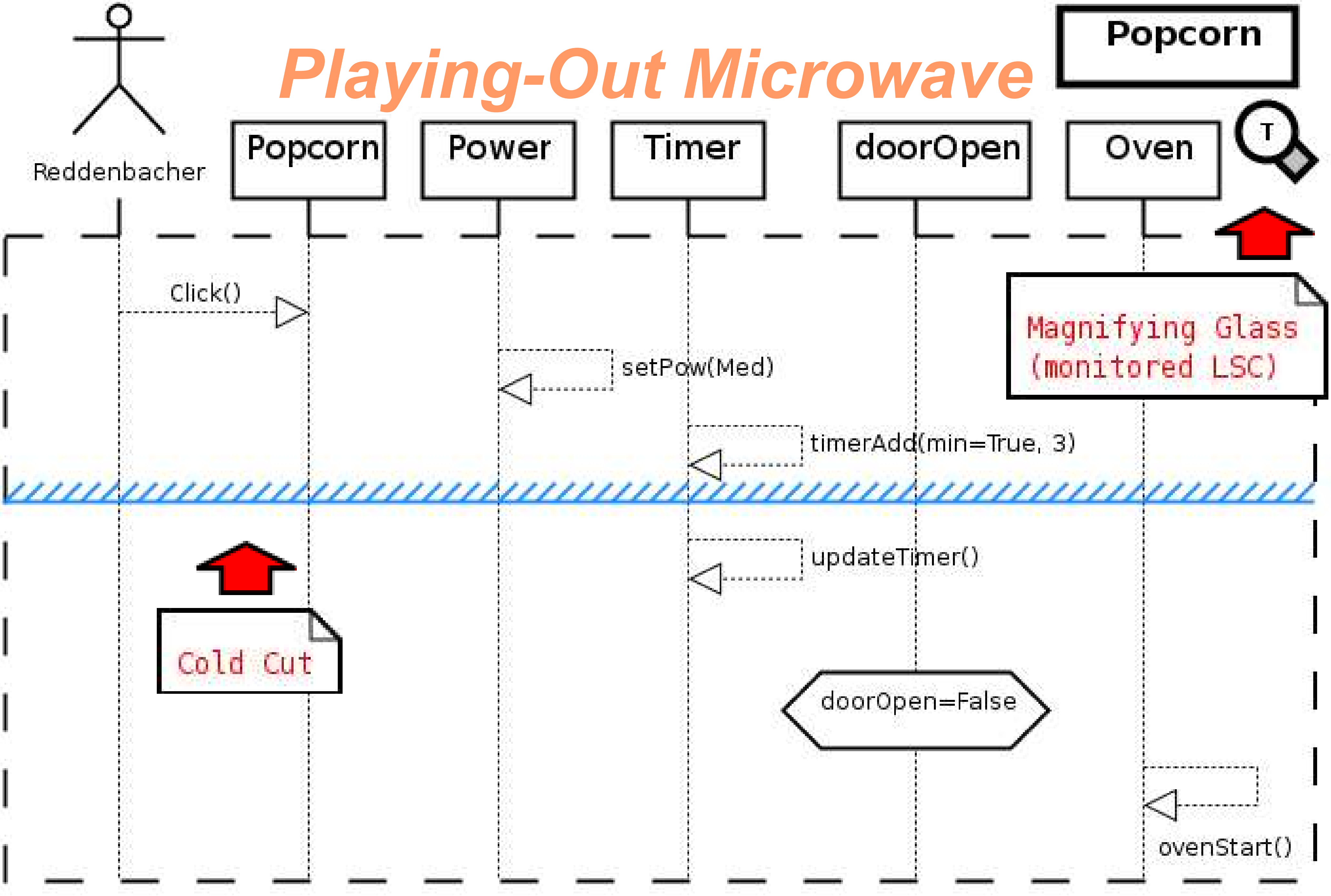
○ Microwave Door is Open

● Microwave Door is Closed

The Brave Little Microwave

0:0

Playing-Out Microwave



Popcorn

Reddenbacher

Popcorn

Power

Timer

doorOpen

Oven

T

Click()

setPow(Med)

timerAdd(min=True, 3)

updateTimer()

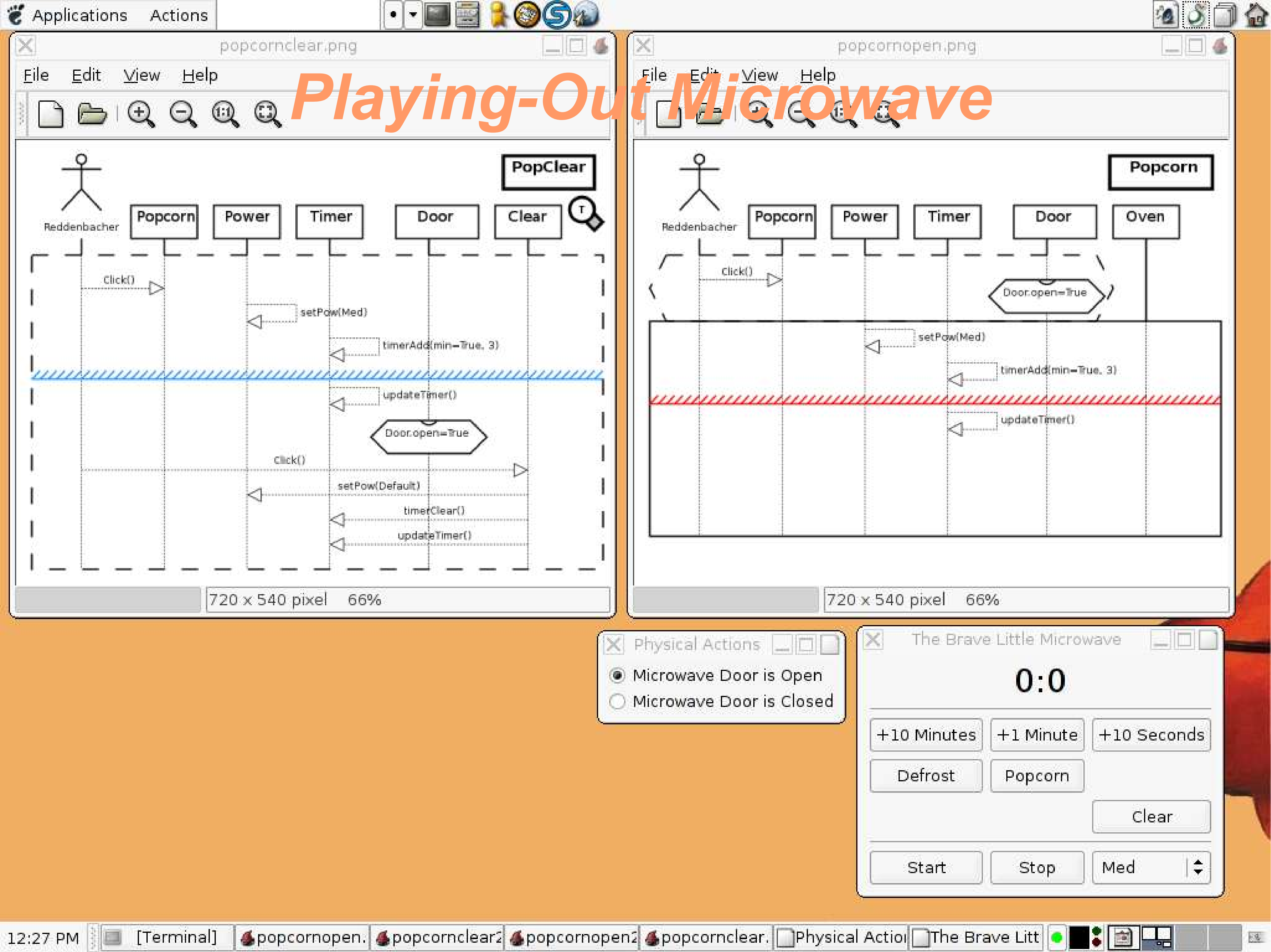
doorOpen=False

ovenStart()

Magnifying Glass (monitored LSC)

Cold Cut

Playing-Out Microwave



Physical Actions

● Microwave Door is Open

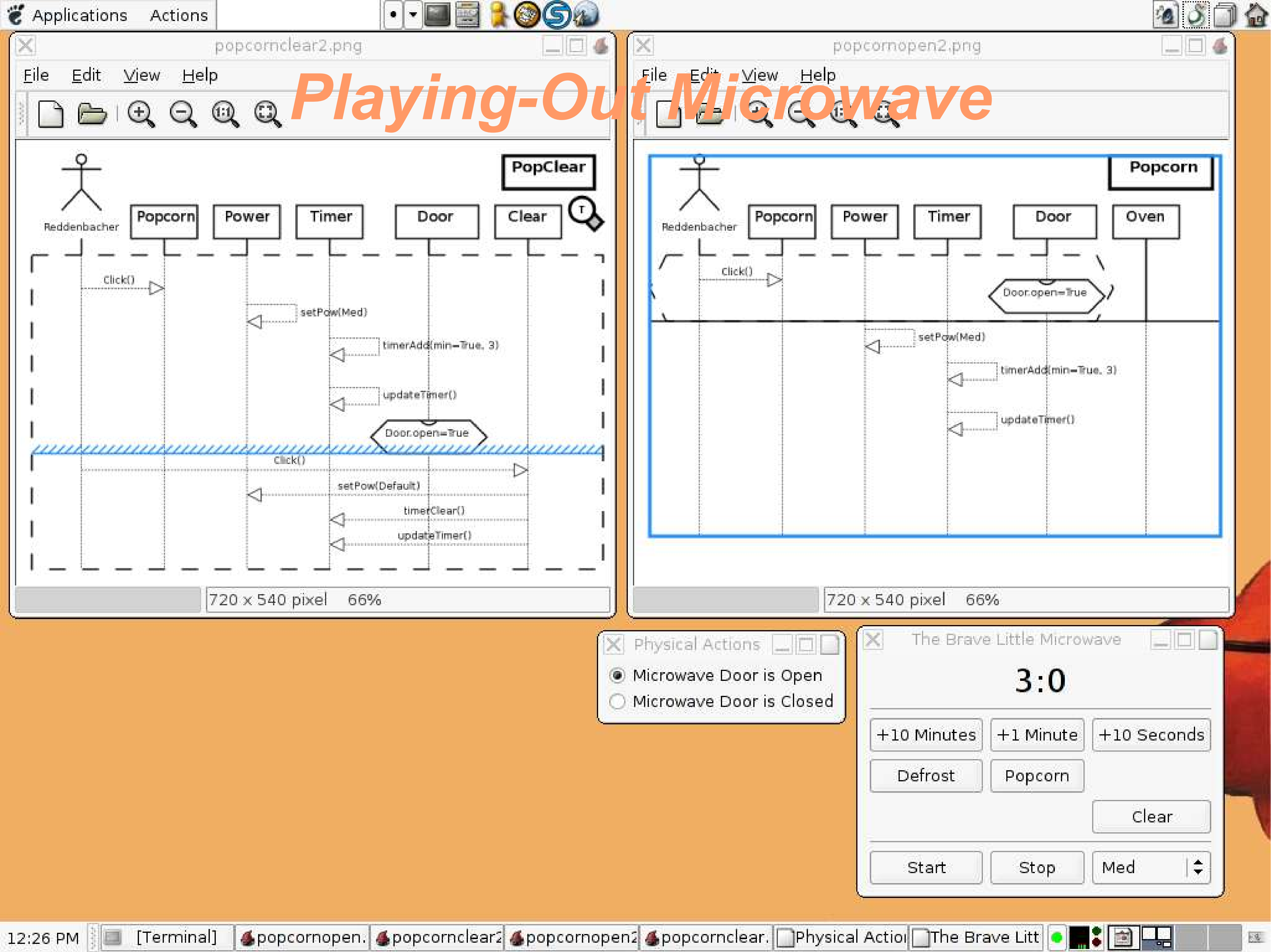
○ Microwave Door is Closed

The Brave Little Microwave

0:0

Med

Playing-Out Microwave



Physical Actions

● Microwave Door is Open

○ Microwave Door is Closed

The Brave Little Microwave

3:0

Med

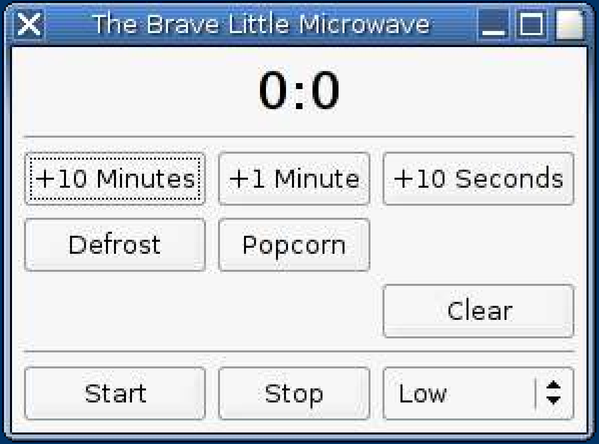
PIPO Conclusions & Questions

The Play-In/Play-Out Approach:

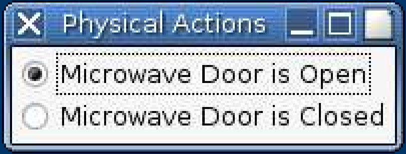
- Simple
- Powerful
- Extendible
- Allows involvement of future users, domain experts

Questions?

LSC to Statechart Transformation



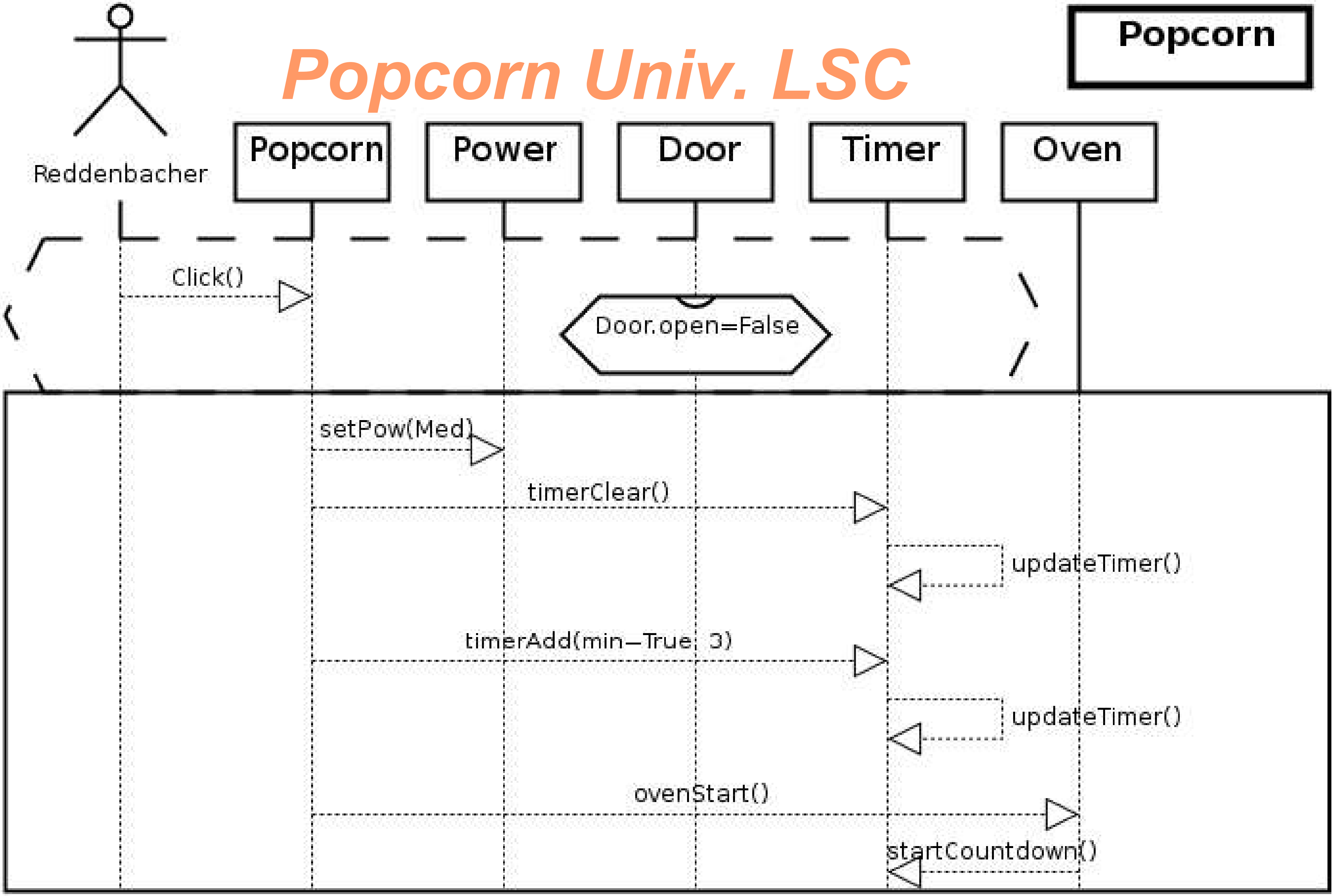
Low



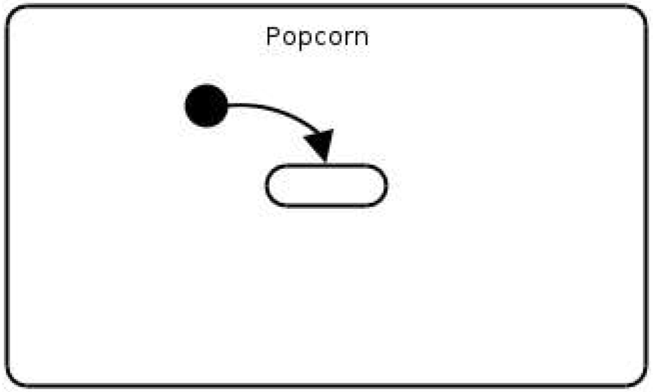
● Microwave Door is Open
○ Microwave Door is Closed

- Goal of this example: transform LSC for the 'Popcorn' button into language of Statecharts
- We'll use multiple Statecharts

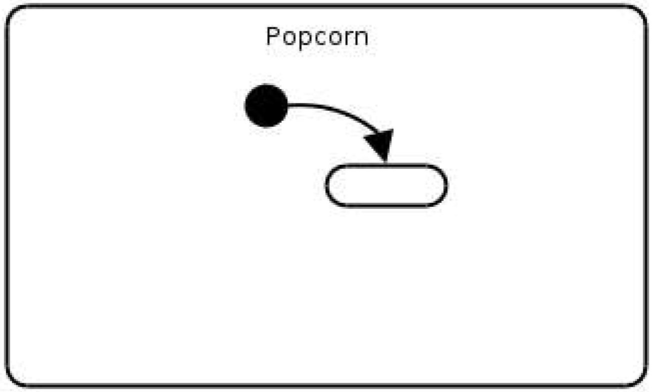
Popcorn Univ. LSC



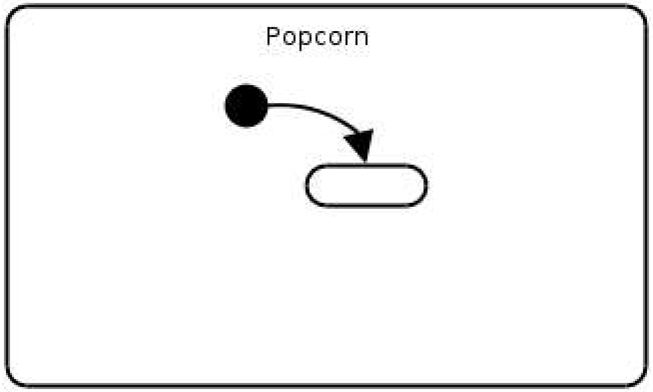
Popcorn



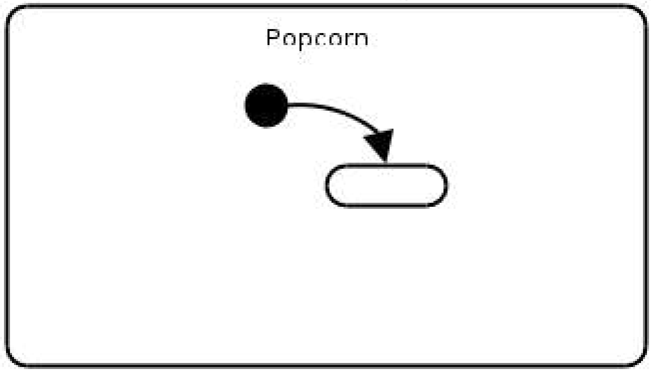
Power



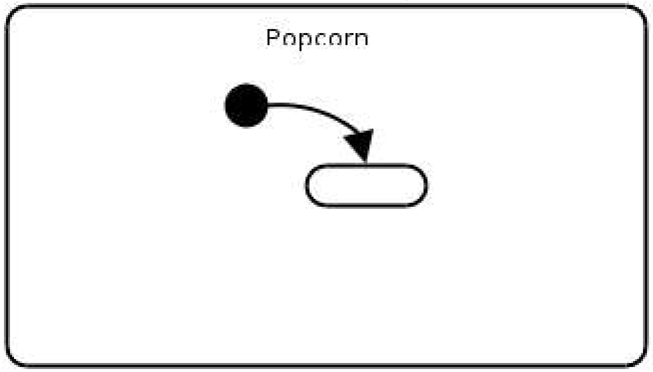
Timer

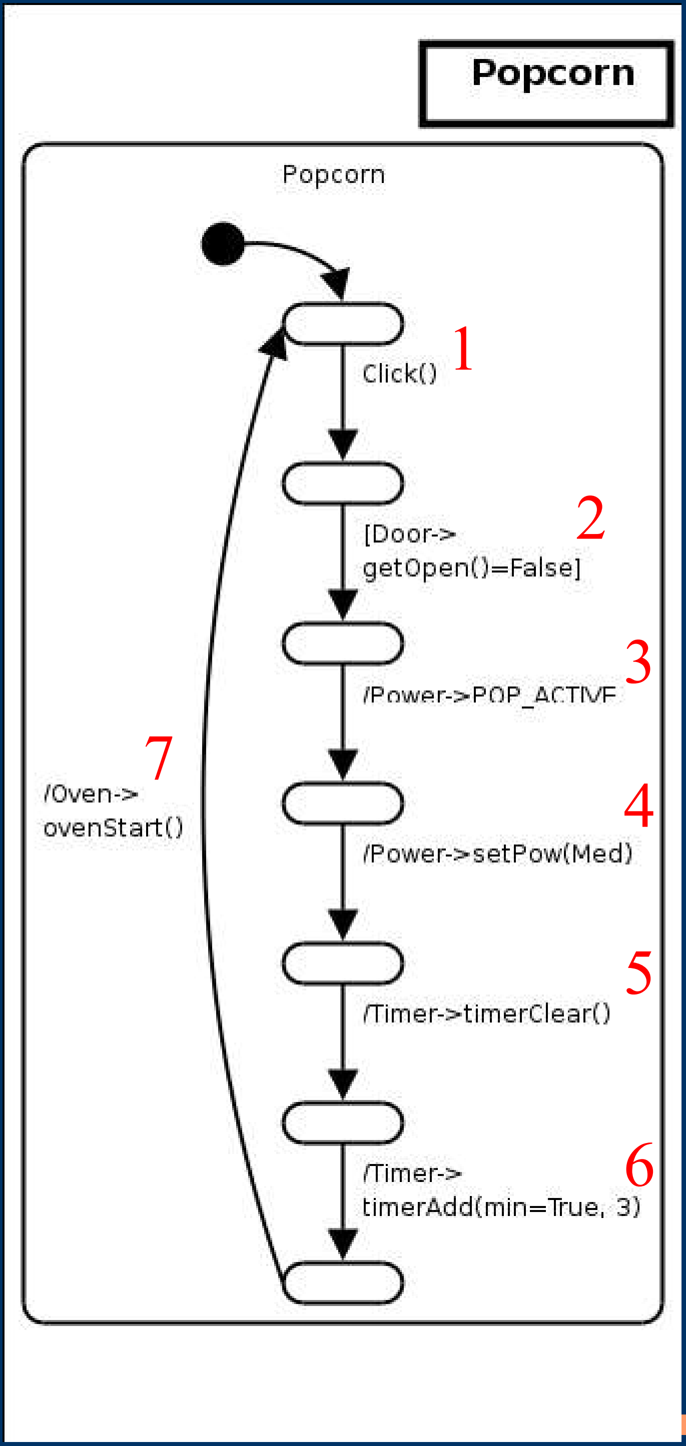


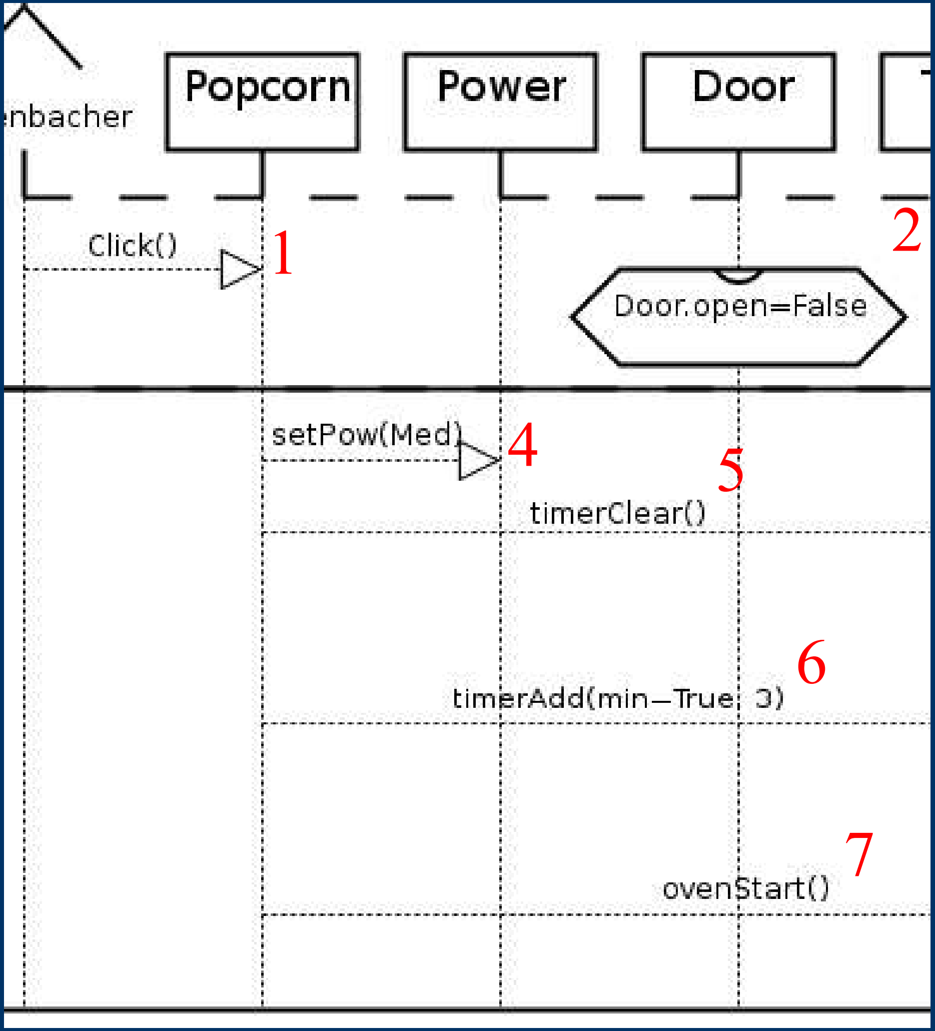
Door



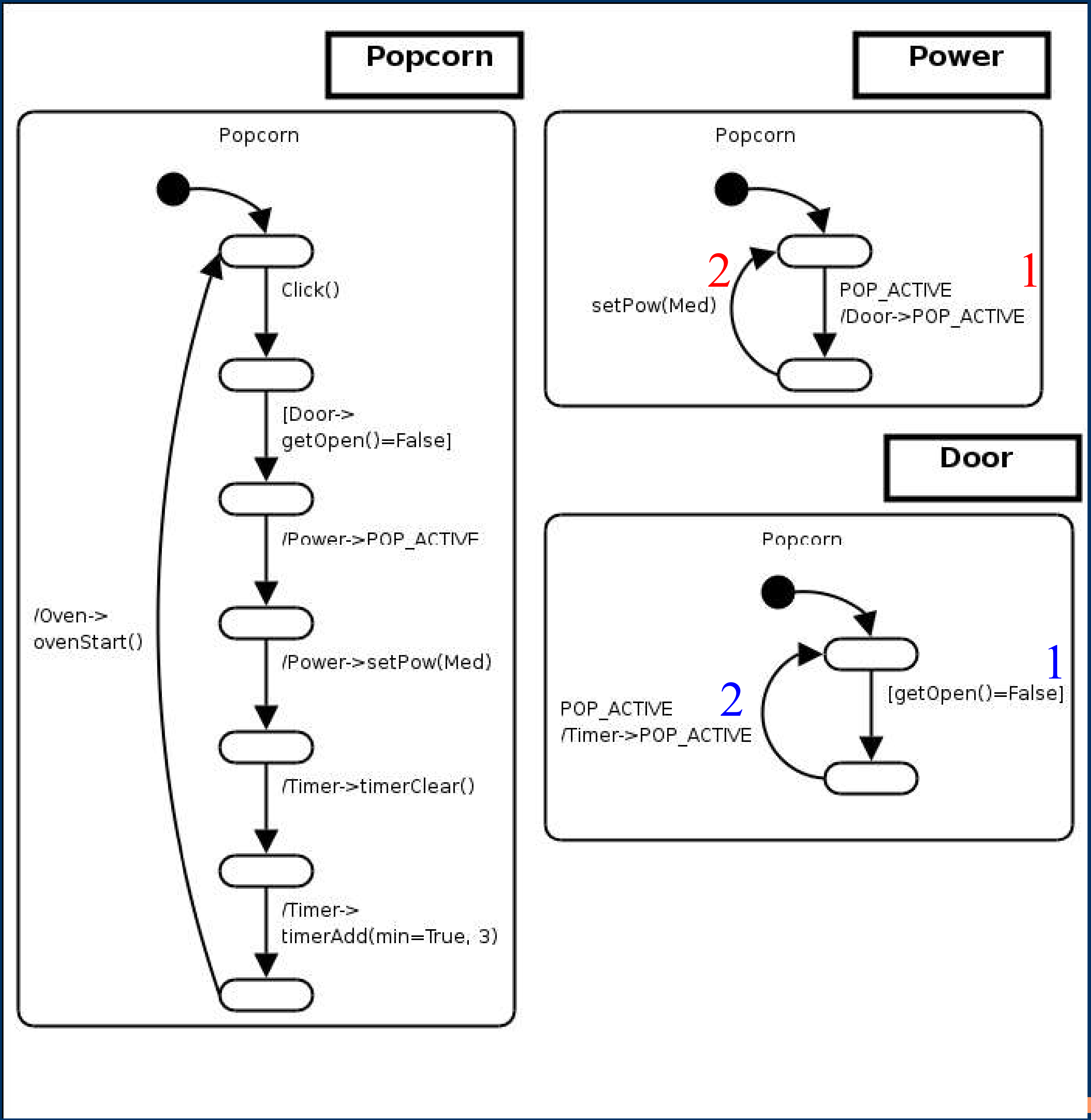
Oven

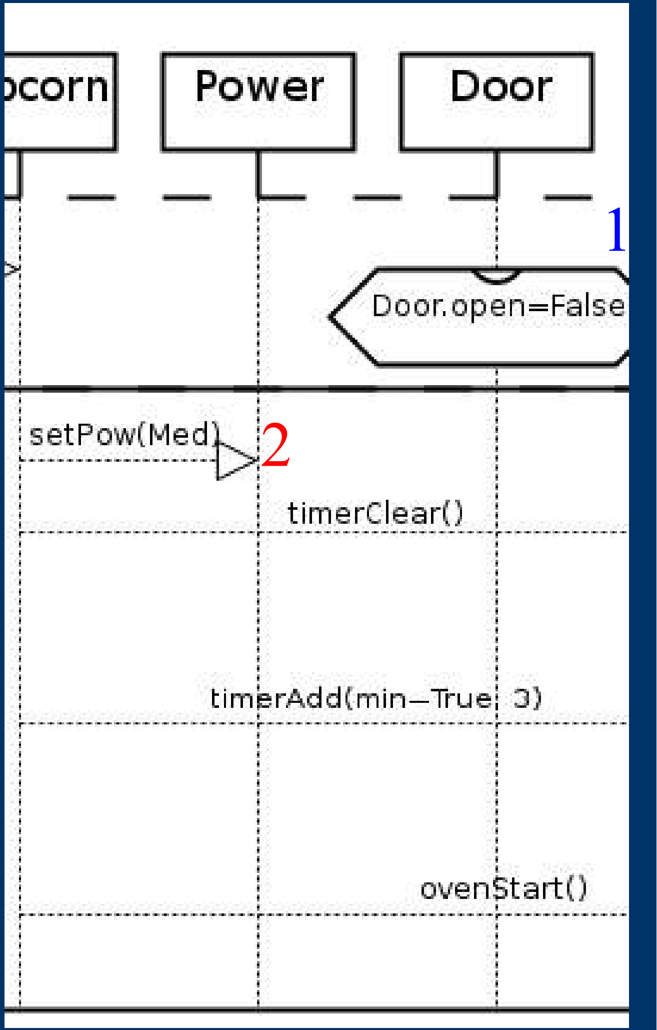




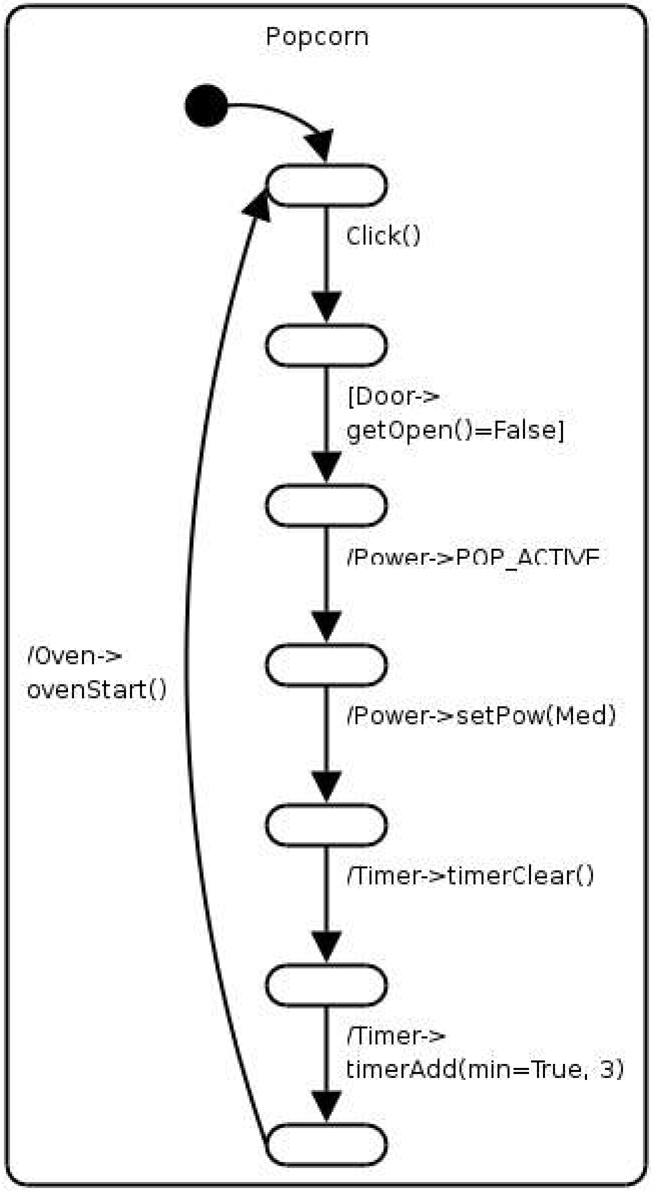


Note transition 3: “/Power->POP_ACTIVE” starts chain of object notification

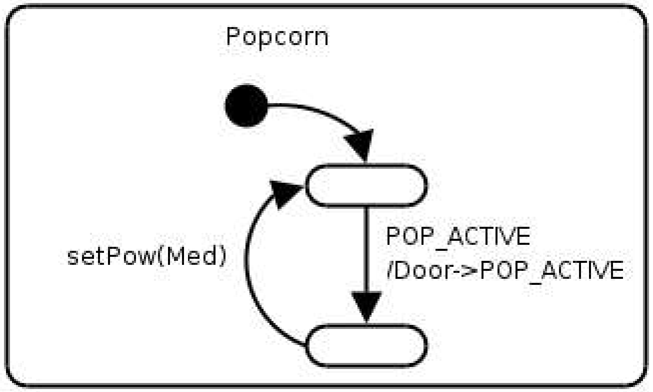

Popcorn
Popcorn
Click()
[Door->
getOpen()=False]
/Power->POP_ACTIVE
/Power->setPow(Med)
/Timer->timerClear()
/Timer->
timerAdd(min=True, 3)
/Oven->
ovenStart()
Power
Popcorn
2
setPow(Med)
1
POP_ACTIVE
/Door->POP_ACTIVE
Door
Popcorn
1
[getOpen()=False]
2
POP_ACTIVE
/Timer->POP_ACTIVE


pcorn
Power
Door
1
Door.open=False
setPow(Med)
2
timerClear()
timerAdd(min=True, 3)
ovenStart()

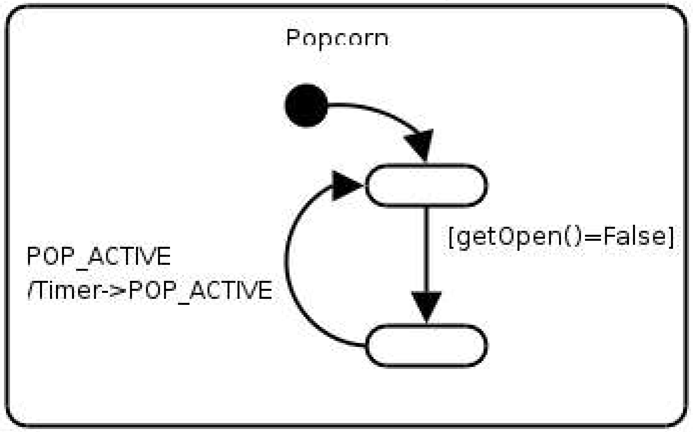
Popcorn



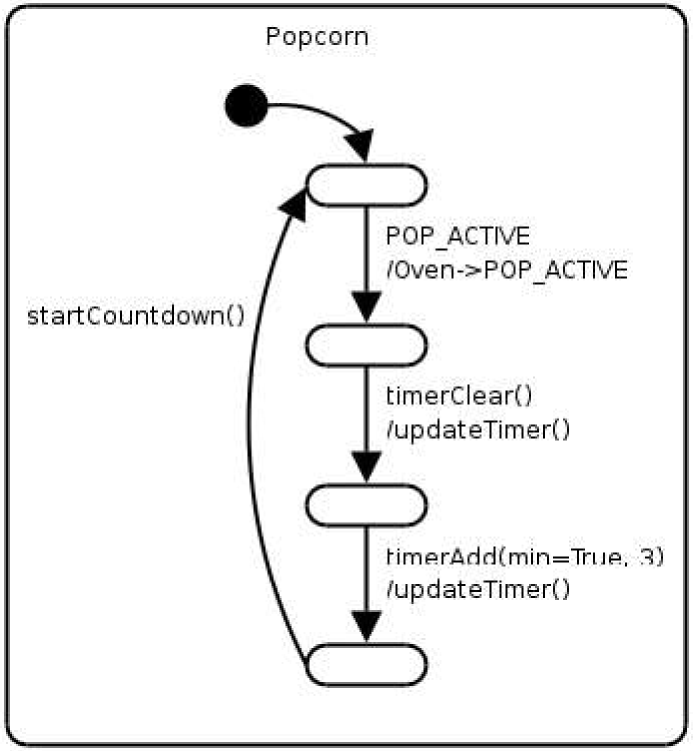
Power



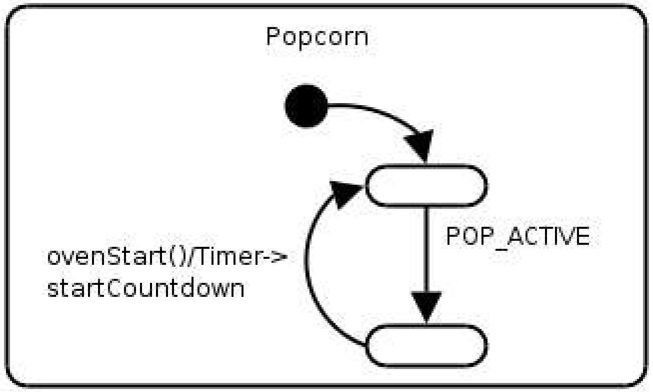
Door



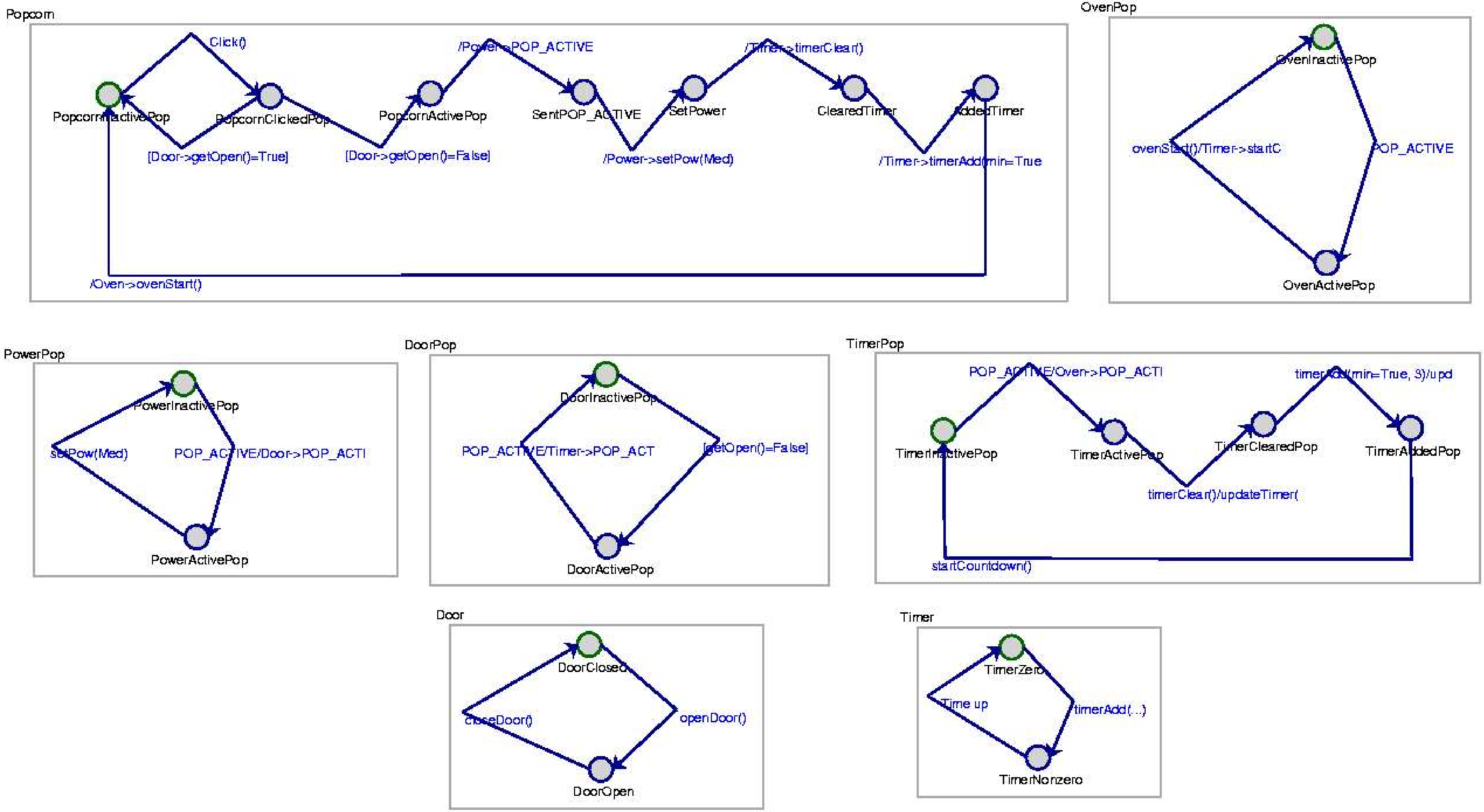
Timer



Oven



Popcorn DChart



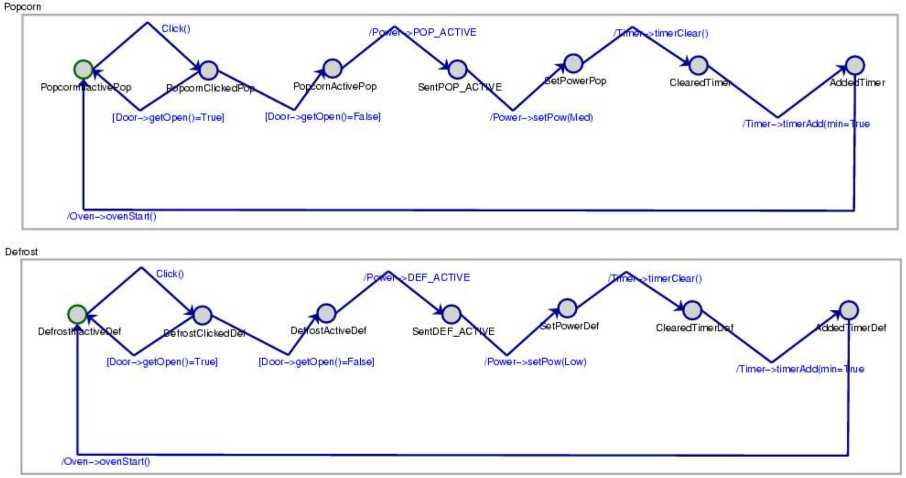
LSC to Statechart Transformation

1. Create one statechart for each unique object in Universal LSC
2. For each statechart:
   1. Create default state
   2. Create one state for each action requiring the object
   3. Chain states together with transitions.
   4. Create one transition from state at end of chain to default state
   5. Label transitions with above actions and "ACTIVE" notification
   6. Create transitions for actions that do not follow PreChart

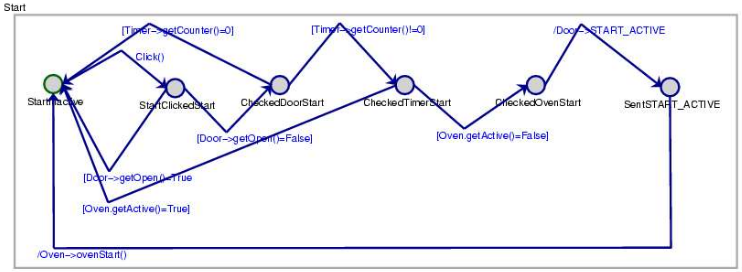
LSC to Statechart Transformation

3. Use orthogonal components if object is in more than one Universal LSC*
4. Check Bad_{max}, set of all supercuts without successors or that lead to those without successors*

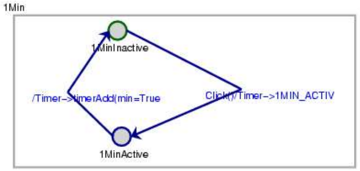
*We didn't do these in the example



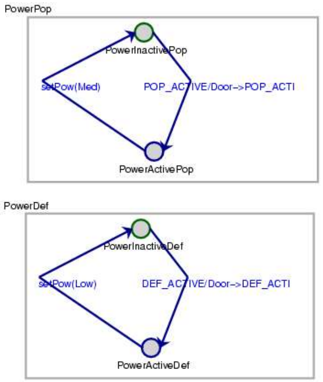
Popcorn, Defrost



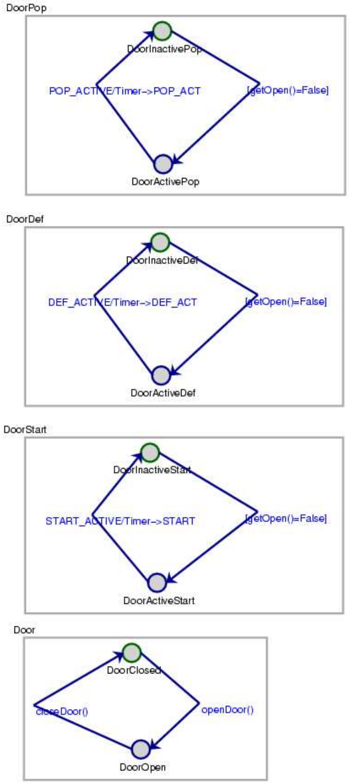
Start



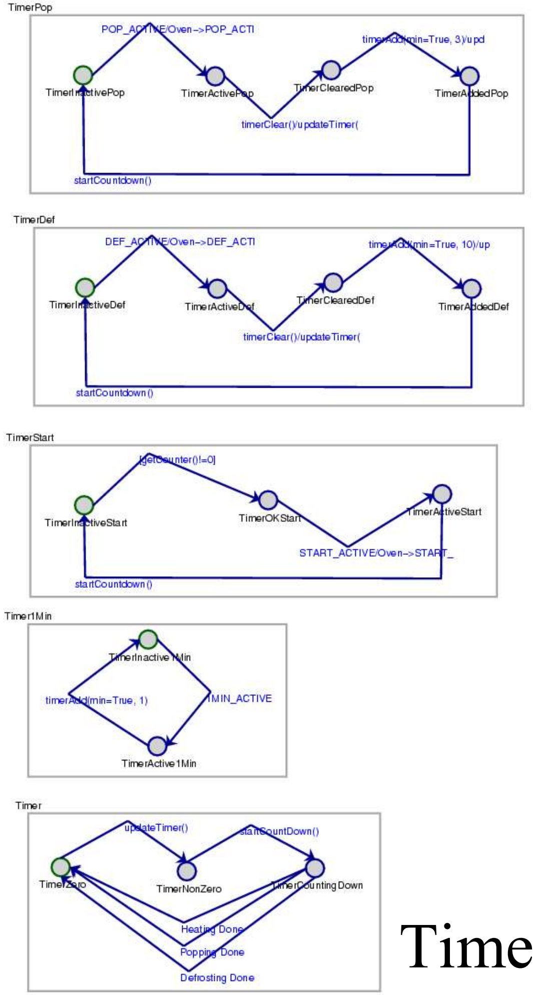
+1Min.



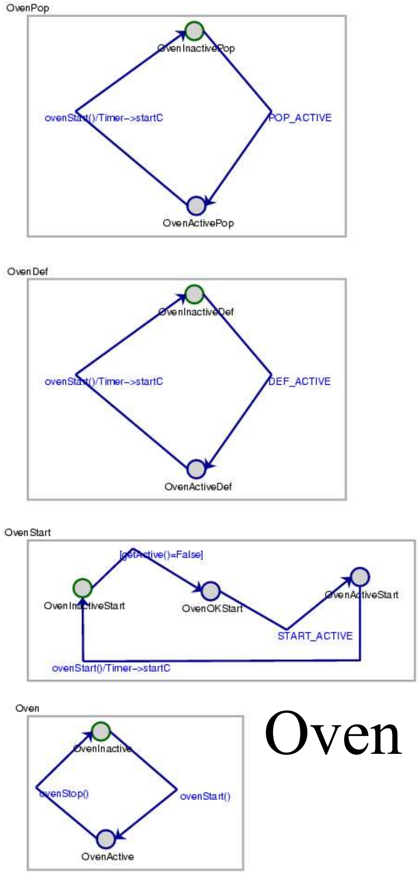
Power



Door



Timer



Oven

LSC Transformation Questions

Final Questions?

References

1. Feng, Thomas Huining. “Charts, a Formalism for Modeling and Simulation Based Design of Reactive Software Systems”. http://moncs.cs.mcgill.ca/people/tfeng/thesis/thesis.html. Feb. 2004.

2. Harel, David. “Can Behavioral Requirements be Executed? (And why would we want to do so?)”

3. Harel, David and Hillel Kugler. “Synthesizing State-Based Object Systems from LSC Specifications”.

4. Harel, David and Rami Marelly. “Specifying and Executing Behavioral Requirements: The Play-In/Play-Out Approach”. September 10, 2002.