The Play-In/Play-Out Approach

Riandi Wiguna
rian.wiguna@mail.mcgill.ca
School of Computer Science
McGill University
March 8, 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. Play-Engine Components
7. Play-Engine Functions
8. Advanced Topics
9. Conclusions & Questions

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, with additional information from:

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

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

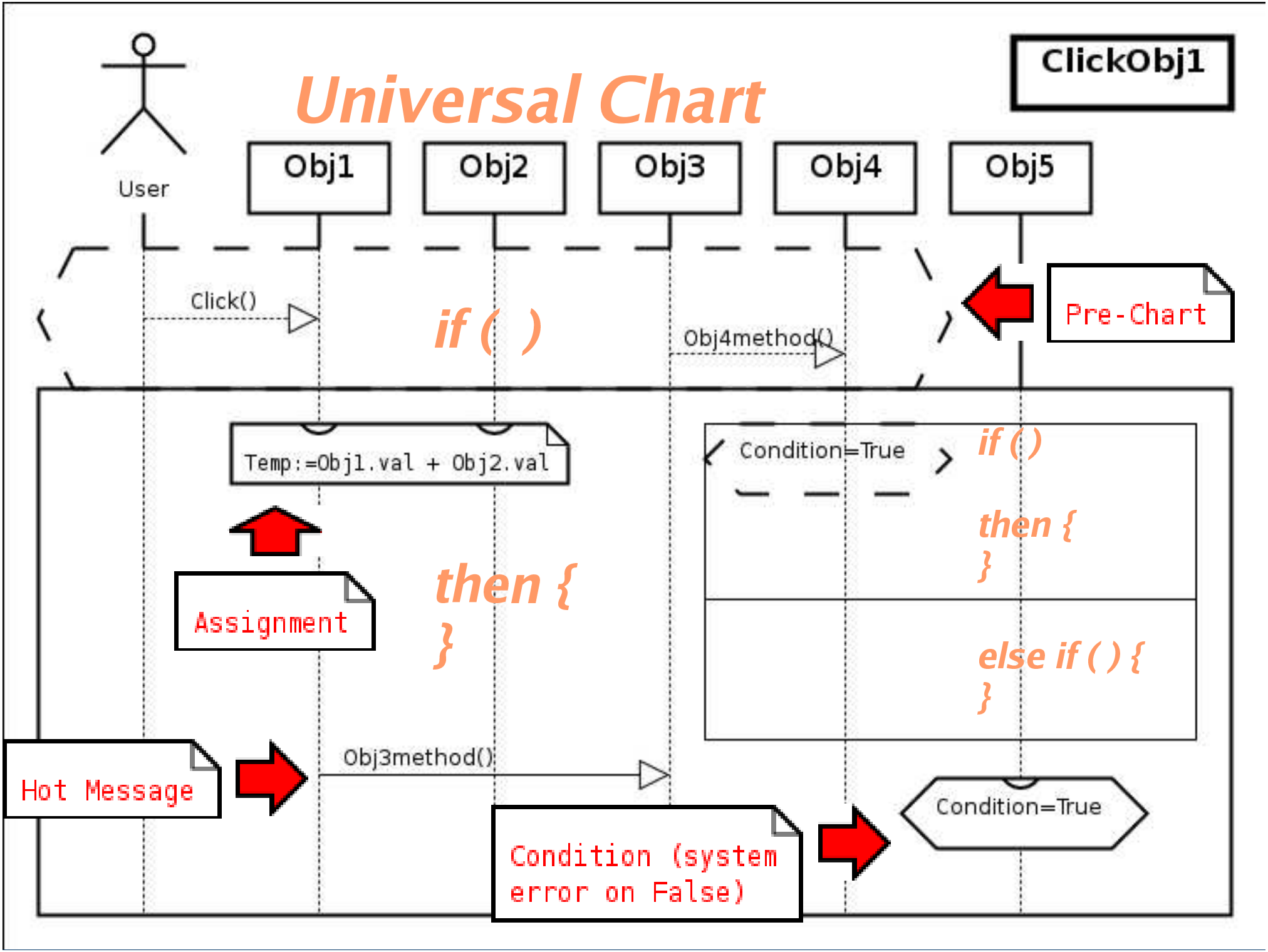
- Model system reactions that *may* happen
- Drawn with dashed border
- Must be able to run to completion in at least one system scenario
- Monitored during Play-Out

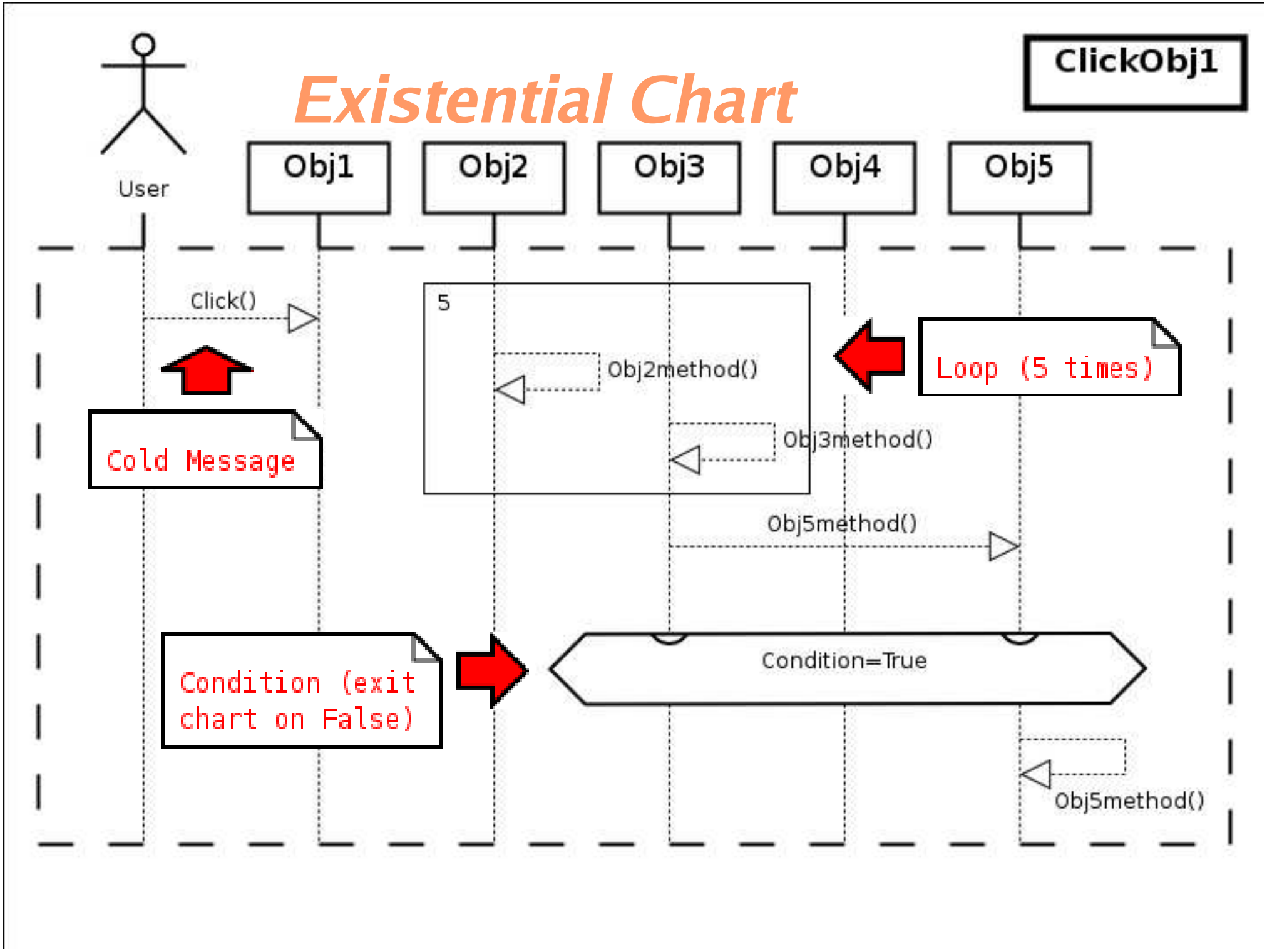
LSC Logic Symbols

- 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
ClickObj1
User
Obj1
Obj2
Obj3
Obj4
Obj5
Click()
if ( )
Obj4method()
Pre-Chart
Temp:=Obj1.val + Obj2.val
Condition=True
if ( )
then {
}
Assignment
then {
}
else if ( ) {
}
Hot Message
Obj3method()
Condition=True
Condition (system error on False)


Existential Chart
ClickObj1
User
Obj1
Obj2
Obj3
Obj4
Obj5
Click()
5
Obj2method()
Obj3method()
Loop (5 times)
Cold Message
Obj5method()
Condition (exit chart on False)
Condition=True
Obj5method()

(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...”)

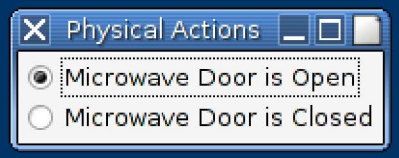
Play-Out

- 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

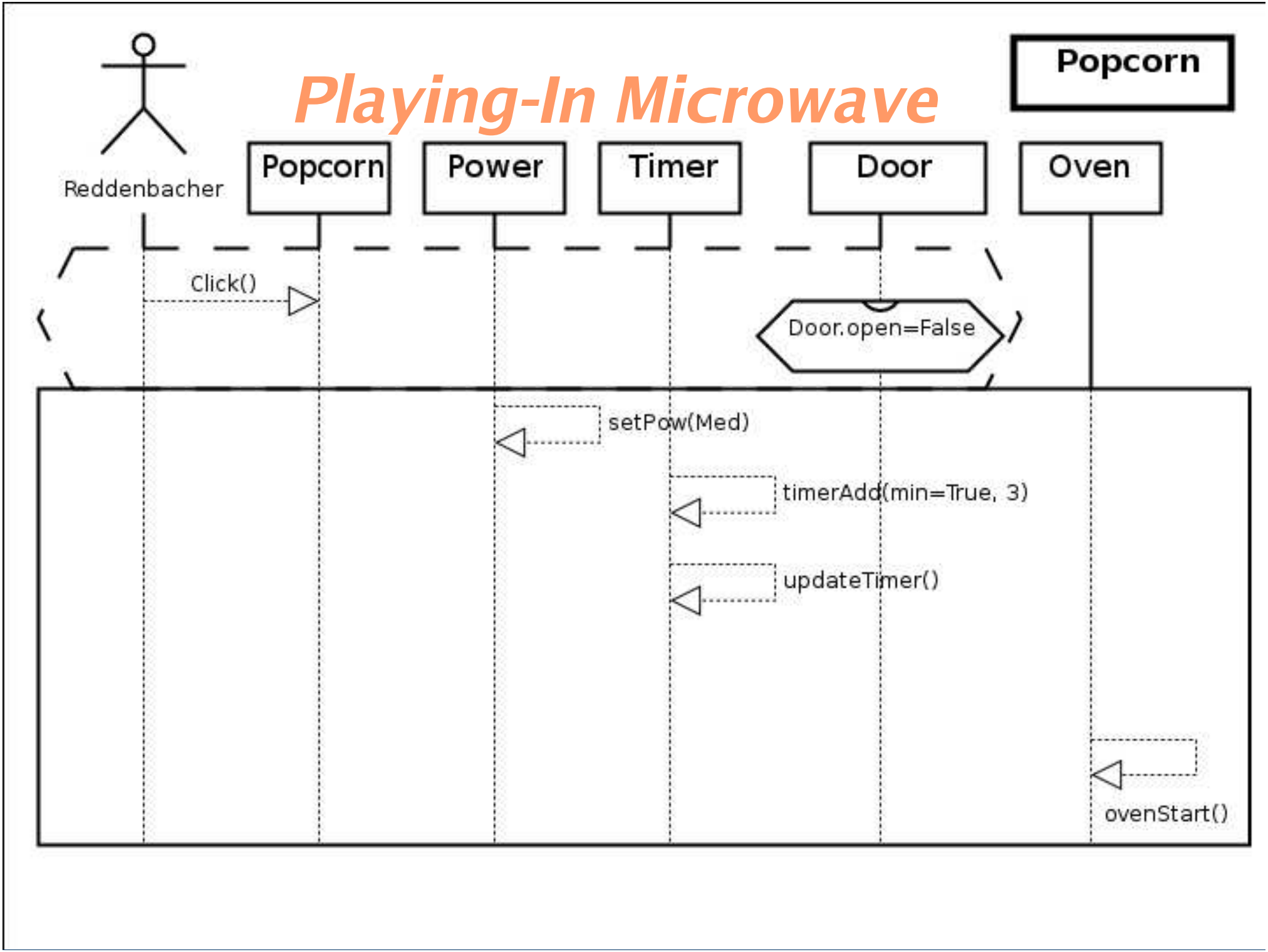
- “Cuts” show position of system in the LSCs
  - Hot (“Combed” red line, system aborts if LSC violated)
  - Cold (“Combed” blue line, system exits LSC if LSC violated)
- Play-Out runs can be saved in XML format
- LSCs are framed in blue when completed
- LSCs are crossed out when violated

Example: Simple Microwave

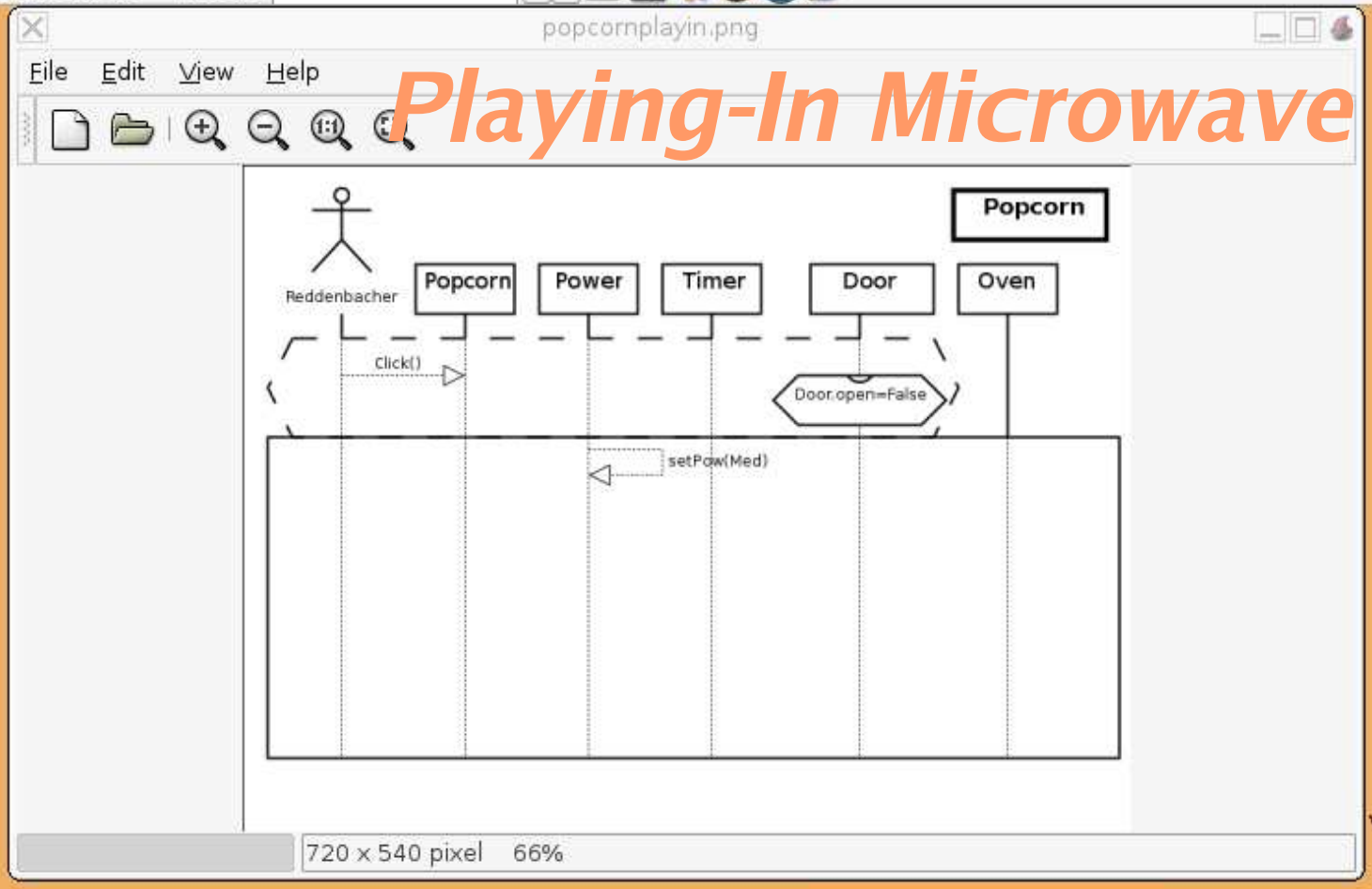


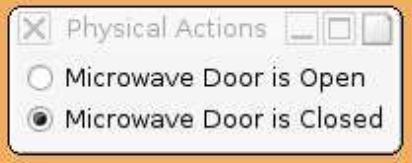
- System Constraints:
  - Take button presses as **"TimeRemaining"**
  - Start microwave on event **"Start"**
  - Stop microwave on any of below
    - event "**Stop**"
    - event "**OpenDoor**"
    - **"TimeRemaining == 0"**
    - event "**SmokeDetected**"

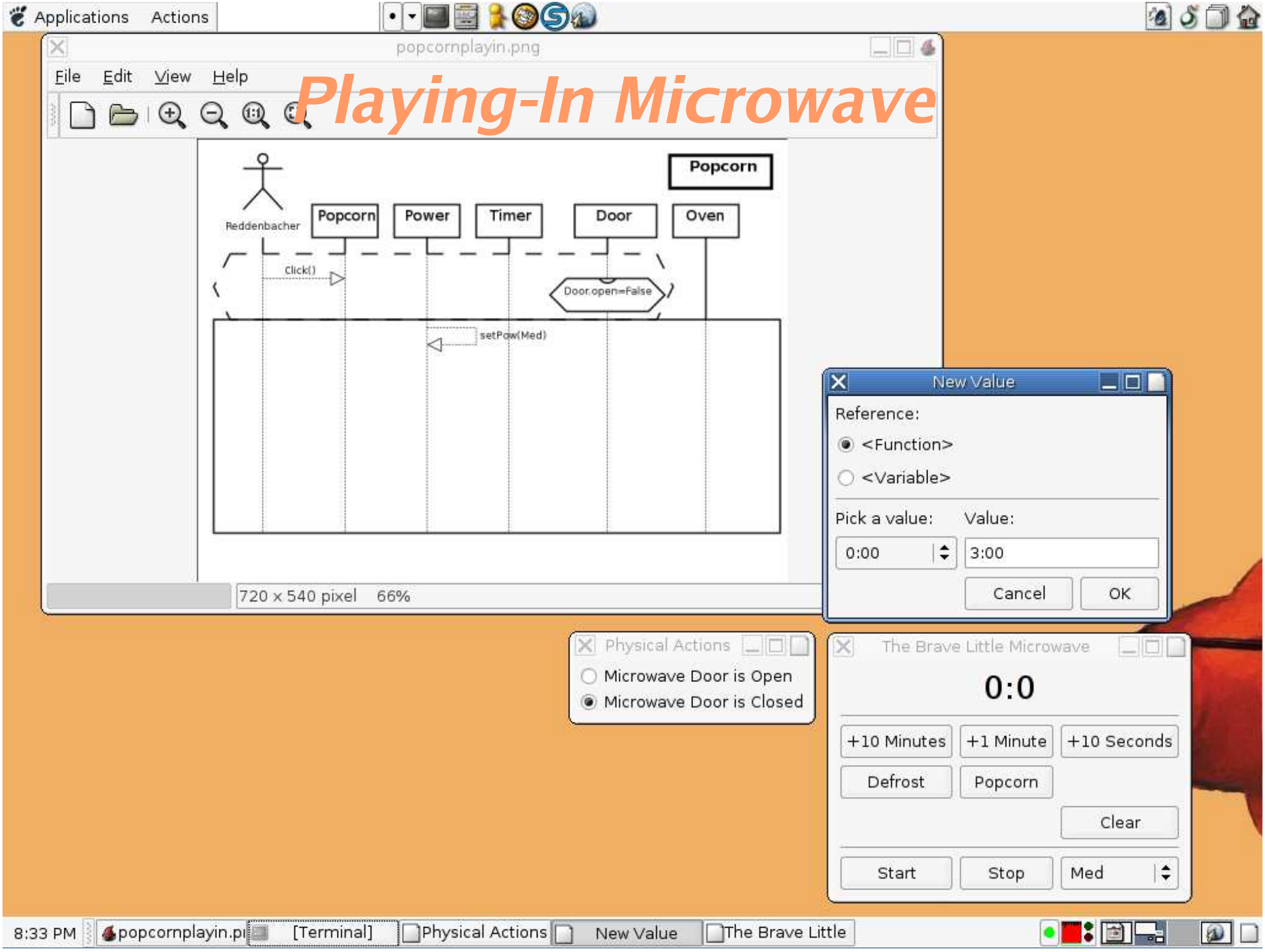
Playing-In Microwave

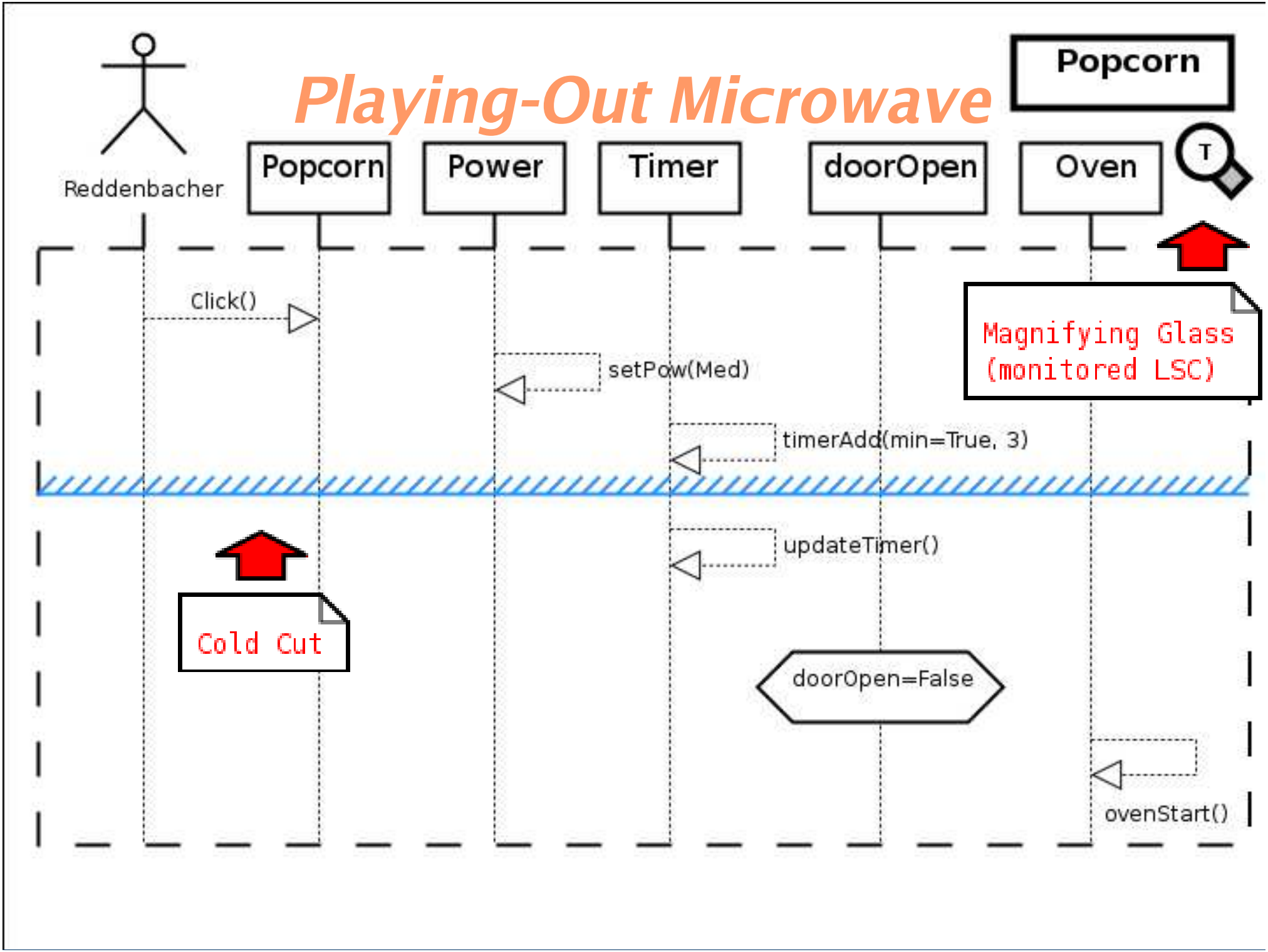


Playing-In Microwave

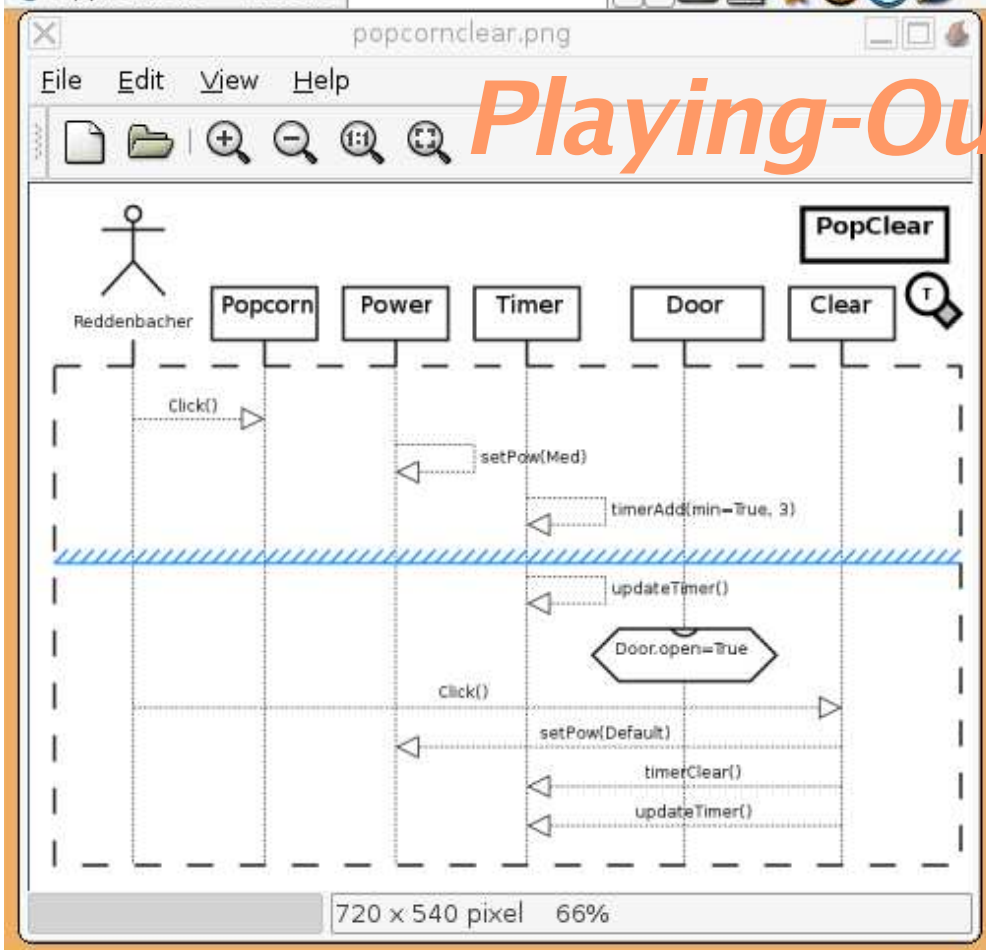


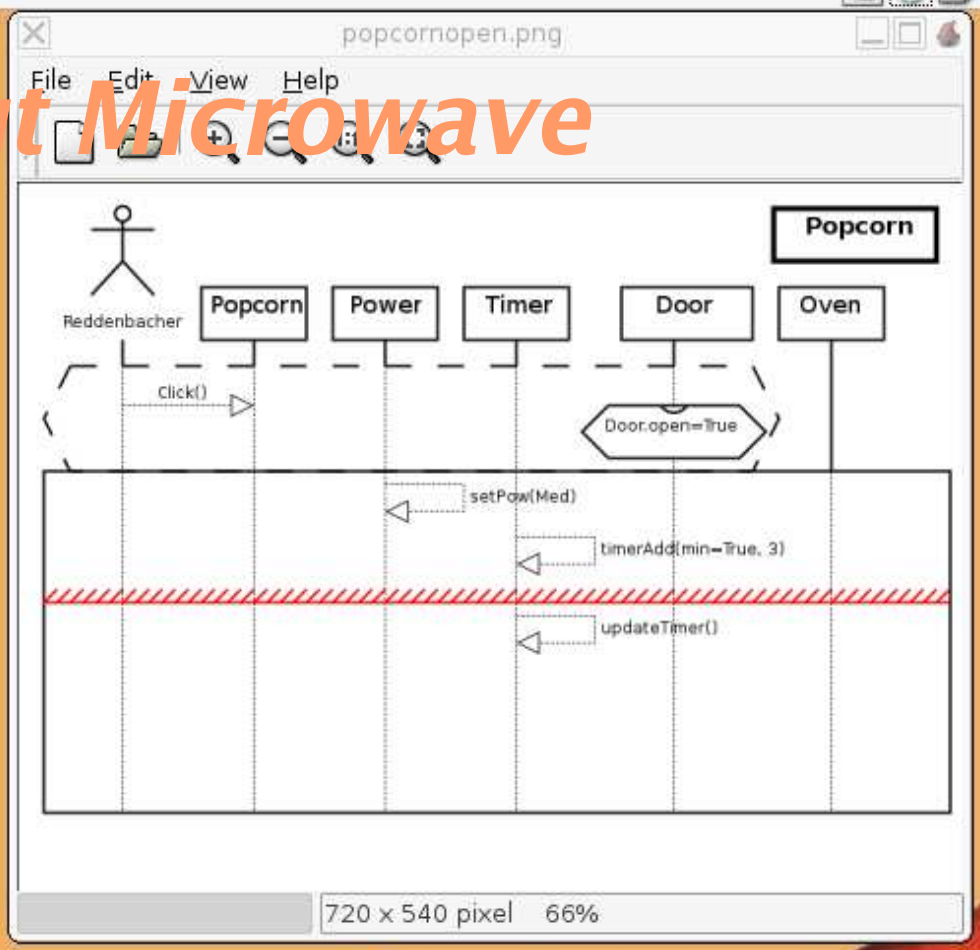


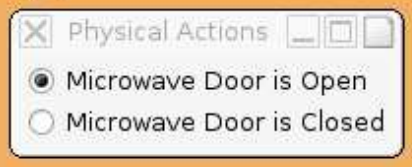

Applications
Actions
popcornplayin.png
File
Edit
View
Help
Playing-In Microwave
Popcorn
Reddenbacher
Popcorn
Power
Timer
Door
Oven
Click()
Door.open=False
setPow(Med)
720 x 540 pixel 66%
New Value
Reference:
<Function>
<Variable>
Pick a value:
Value:
0:00
3:00
Cancel
OK
Physical Actions
Microwave Door is Open
Microwave Door is Closed
The Brave Little Microwave
0:0
+10 Minutes
+1 Minute
+10 Seconds
Defrost
Popcorn
Clear
Start
Stop
Med
8:33 PM
popcornplayin.pi
[Terminal]
Physical Actions
New Value
The Brave Little


Playing-Out Microwave
Popcorn
Reddenbacher
Popcorn
Power
Timer
doorOpen
Oven
T
Click()
setPow(Med)
timerAdd(min=True, 3)
Magnifying Glass
(monitored LSC)
updateTimer()
Cold Cut
doorOpen=False
ovenStart()

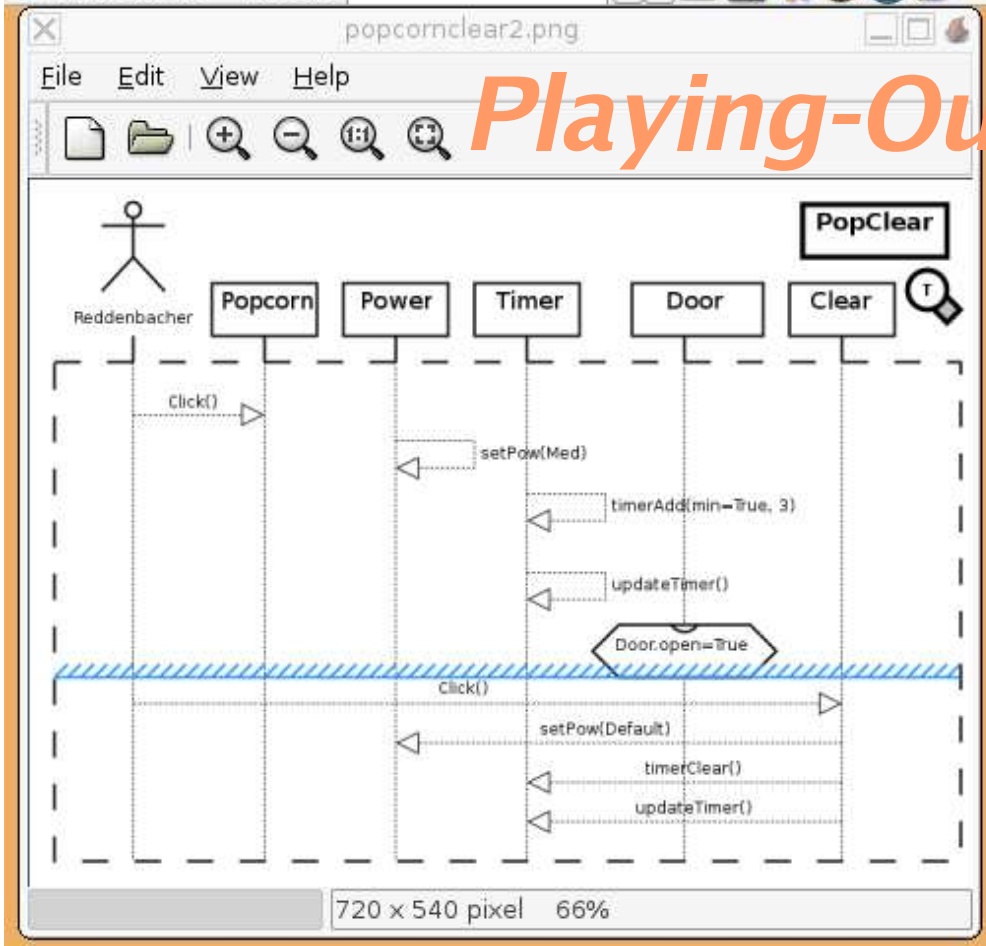
Playing-Out Microwave

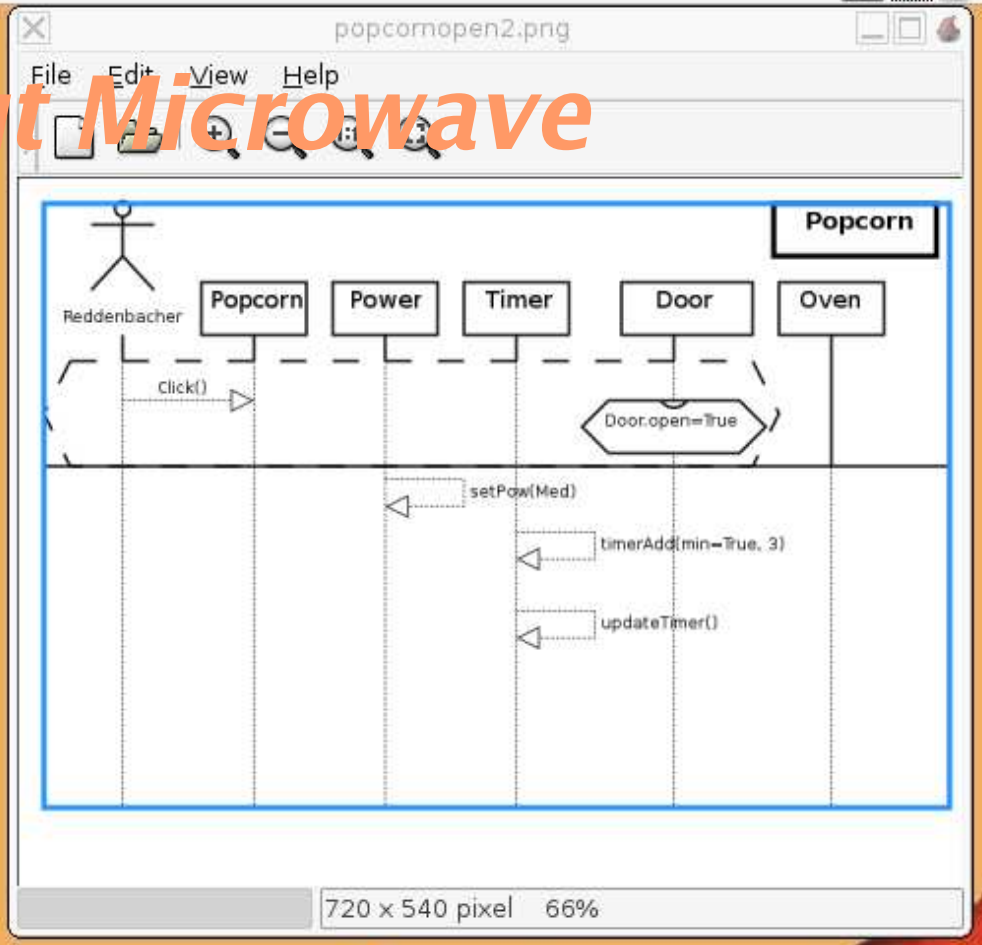


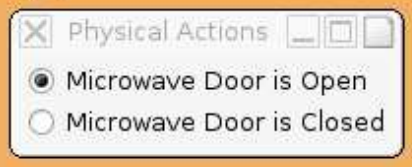




Playing-Out Microwave







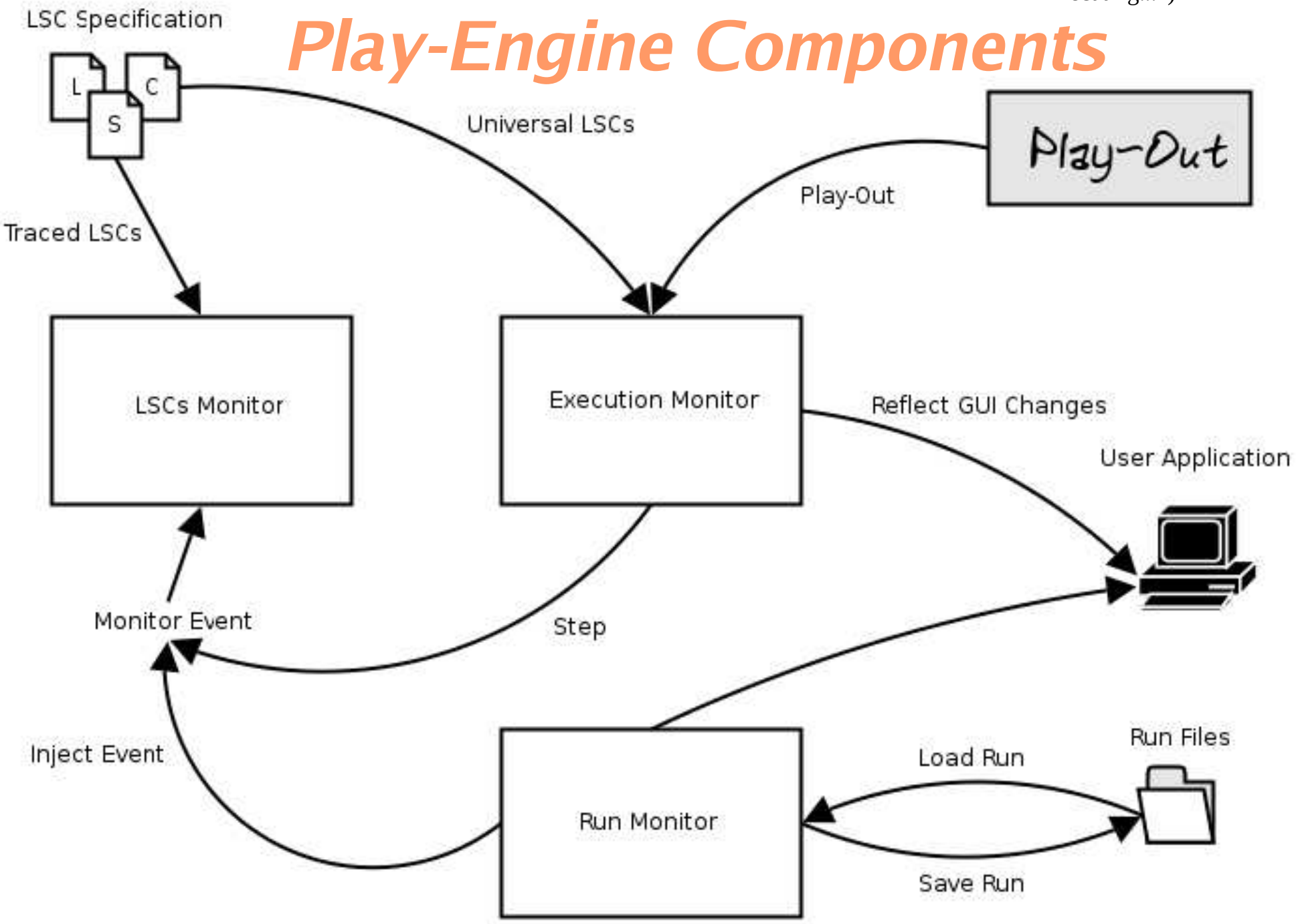
(from pgs. 25-26 of “Specifying and Executing...”)

Play-Engine Components

- Execution Manager
  - Sends data to LSCs Monitor and GUI, to update LSCs and interface
  - Play-In
    - Handles user actions
  - Play-Out
    - Activates universal LSCs
    - Makes system reactions according to universal LSCs
- Run Manager
  - Saves and loads runs
  - Sends data to LSCs Monitor and GUI, to update LSCs and interface
- LSCs Monitor
  - Handles active and other monitored LSCs

(from pg. 26 of “Specifying and Executing...”)

Play-Engine Components



(from pgs. 26-28 of “Specifying and Executing...”)

Play-Engine Functions

- Unify Messages
  - Positive Unification (Messages that can be done simultaneously)
  - Negative Unification (Messages that violate LSC if done simultaneously)
- Get Next Cut
- Find Unifiable Event
- Minimal Event in Chart
- Is Violating Event
- Choose Step

(from pgs. 30-36 of “Specifying and Executing...”)

Advanced Topics

- Multiple Instances
  - Group/class object variables in LSCs
- Non-Deterministic Choice
  - Probabilities in if-else blocks ('Select(25, 75)' => 25% probability of True) in LSCs
- Time and Real-Time Variables
  - Clock object variables in LSCs
  - Play-In shows connections between two objects with time dependencies in LSCs

Advanced Topics

- External Objects
  - Outside object variables in LSCs
- Forbidden Elements
  - Hot or Cold, similar to messages and cuts
  - Play-Out shows connections between forbidden elements and objects in LSCs
- Smart Play-Out
  - Enhances Play-Engine
  - Searches for “correct” super-steps, those that do not violate any universal LSCs
  - Finds a sequence of reactions that leads to a state of zero active universal LSCs

Conclusions & Questions

The Play-In/Play-Out Approach:

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

Final Questions?

References

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

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

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