Rewriting Logic Approach to Separating Policy Rules from Behavioral Specification

Xiaoxi DONG and Shin NAKAJIMA
National Institute of Informatics
Japan
2010 May

Content

- Background
- Issue and Our Approach
- Policy Enforcer
- Maude
- A Brief Introduction to the Guiding System
- Guiding System and Policy
- Test and Result
- Future Work

Background

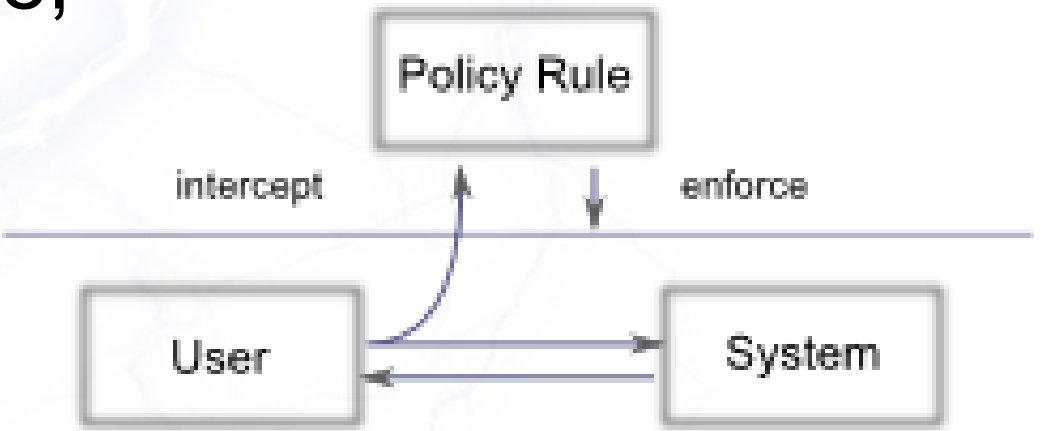
- Software systems involving users are prone to unexpected user behavior.
- The system may be encoded with rules to ensure the inputs are expected user behavior.
- However, the rules may evolve over time. It results in updating the entire software.
- Encoding the rules in the system results in high maintenance cost and bad extensibility.

Issue and Our Approach

- The pamphlet can not force the users to behave properly.
- Unexpected user behavior could result in serious problem.
- We present a prototype of a two-tiered framework that separates the policy rules and the rest of the system.

Policy Enforcer

- A policy enforcer has a set of policy rules.
- Policy rules could be,
  - Stop rule
  - Correction rule
  - Allow rule



- The user behavior is sent to the policy enforcer and it is sent back to lower level if it is proper.

Maude

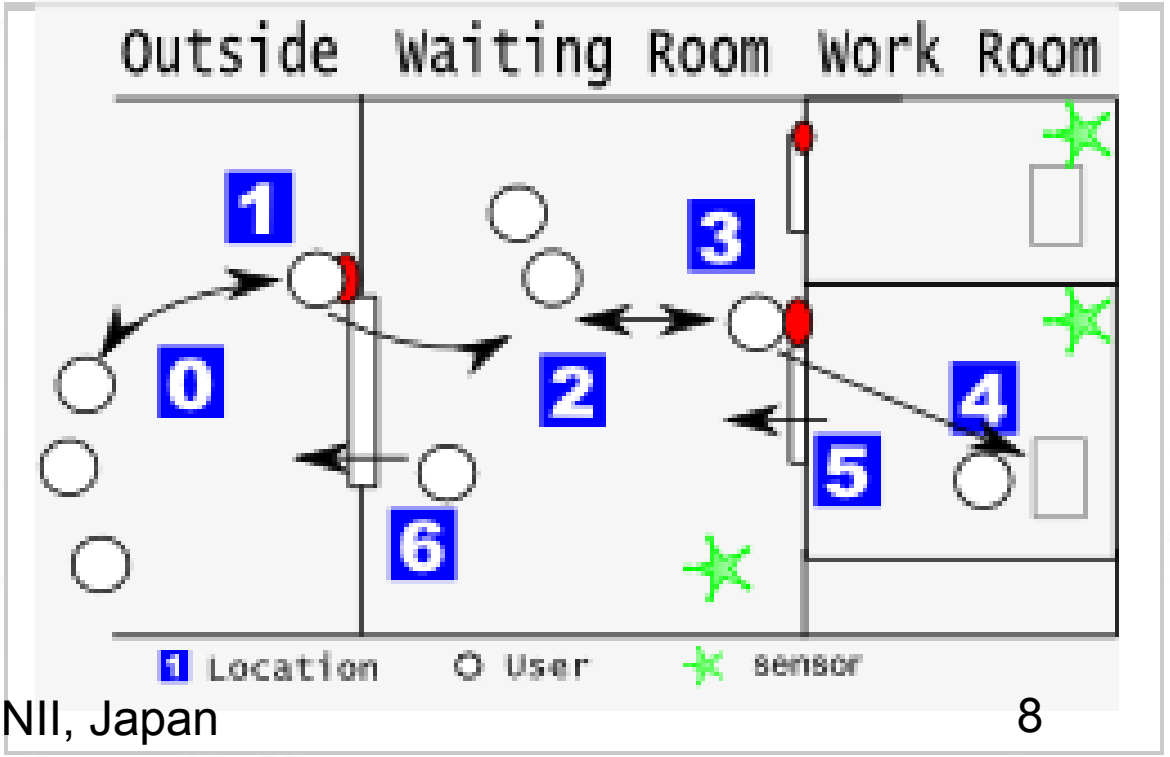
- Maude is a high-performance reflective language and system.
- It is influenced by the OBJ3 language family.
- Maude can support
  - Static: equational logic
  - Dynamic: rewriting logic
- Maude has useful analysis functionality, such as reduce, rewrite, search and model check.
- We use Core Maude in this project.

Content

- Background
- Issue and Our Approach
- Policy Enforcer
- Maude
- **A Brief Introduction to the Guiding System**
- Guiding System and Policy
- Test and Result
- Future Work

Intro. to the Guiding System

- We use the example of a guiding system to demonstrate the use of the two-tiered framework.
- A Guiding system example

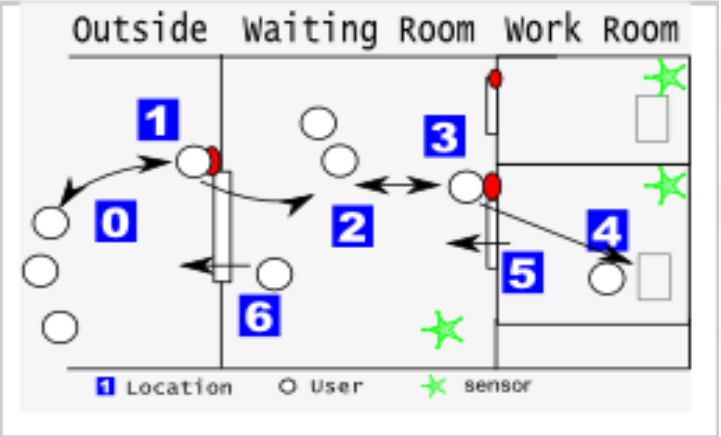


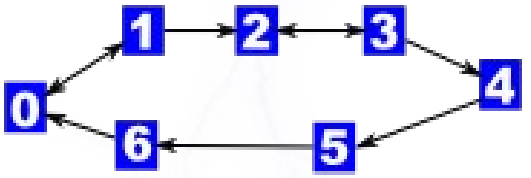
Guiding System in Maude

- Summary
  - Location Graph
  - Breakdown the Guiding System
  - Extended Mealy Machine
  - The User Machine
  - Guiding System
  - Enforce the Policy
  - Guiding System and Policy

Location Graph

- At the very basic level, we can abstract the guiding system a directed graph

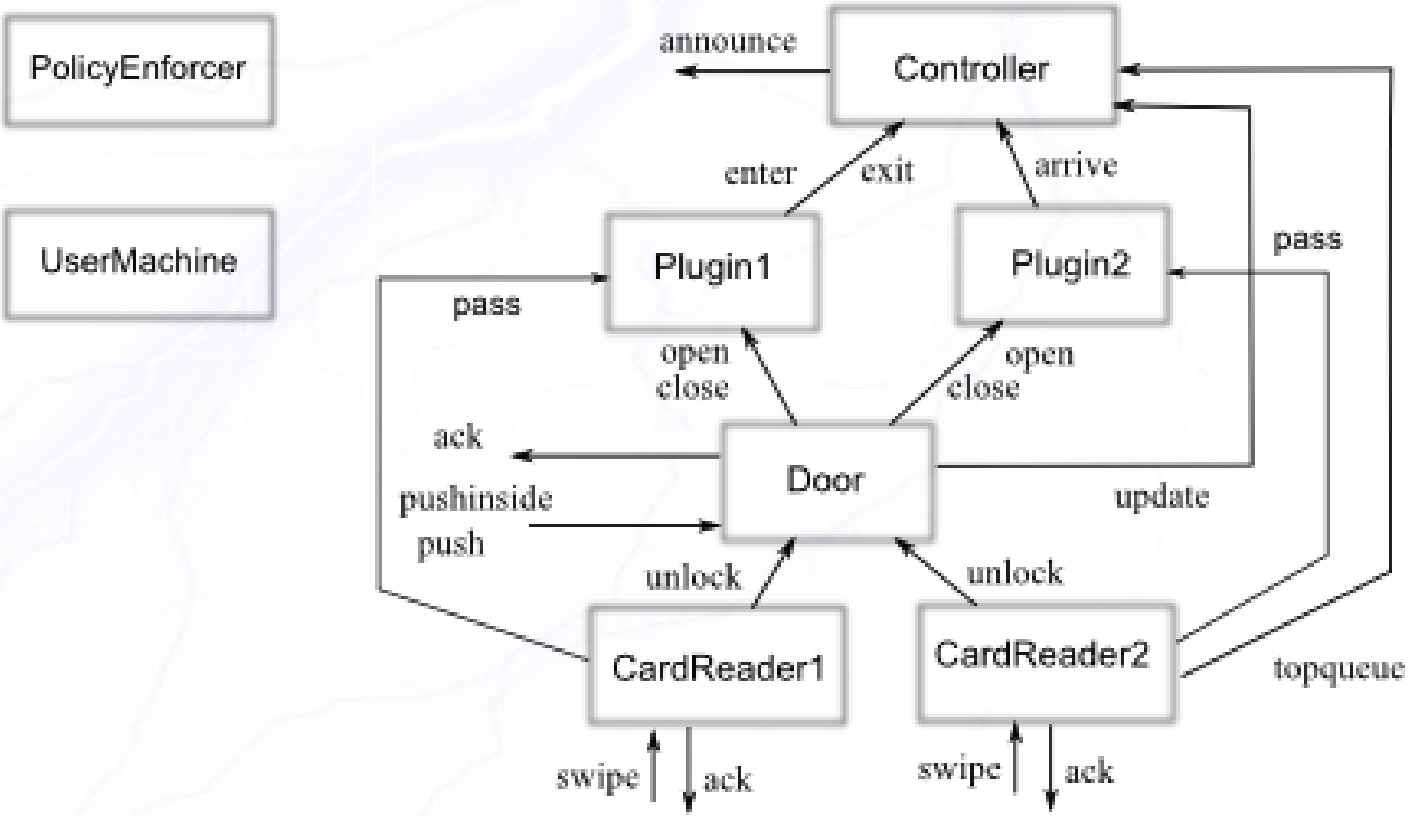




- This graph comprises of
  - Node – Clients' location in the system
  - Direct Edge – Clients moves along the edges
  - Node Capacity – The maximum number of clients in each location

Guiding System Breakdown

- The guiding system consists of the following components. Each of them is a Mealy machine.



Extended Mealy Machine

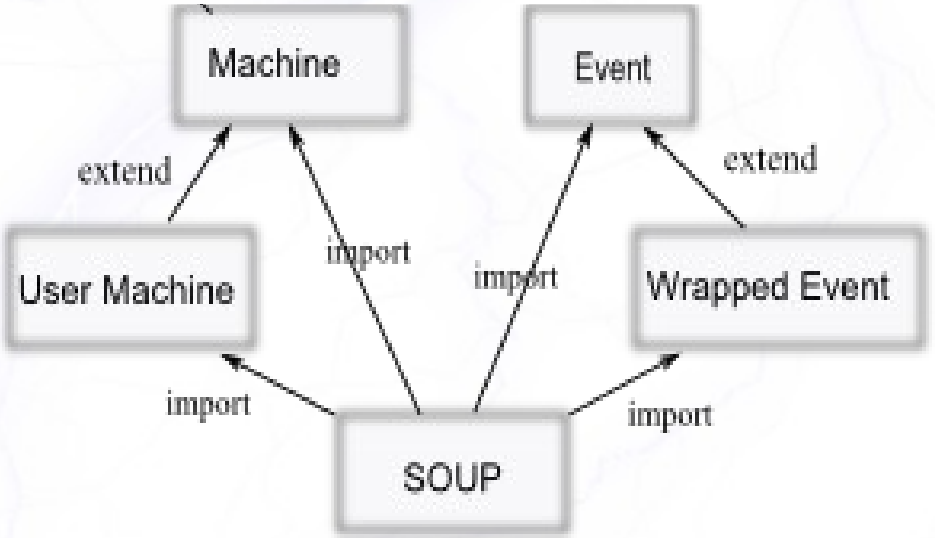
- The components of a guiding system are all extended mealy machines. They has attributes, and communicate by sending and receiving events

$$M = (Q, \Sigma, \rho, \delta, q_0, \mathcal{F})$$

- The machine states change because of the events
- We add a WrappedEvent for the policy enforcer.
- We add a UserMachine to make simulation faster.

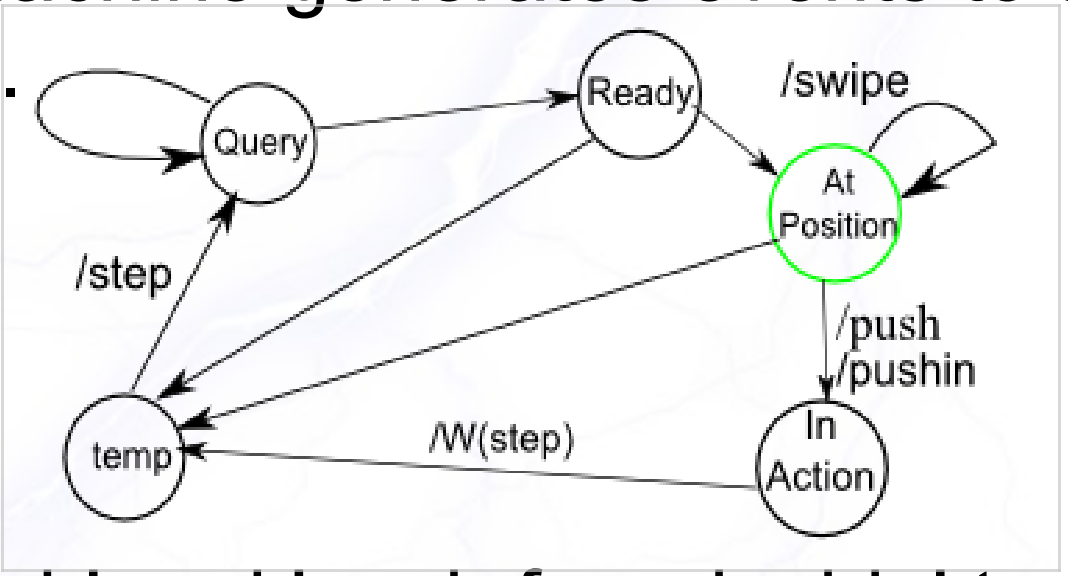
Extended Mealy Machine

- The pool of Machine, UserMachine, Event and WrappedEvent makes a SOUP.



The UserMachine

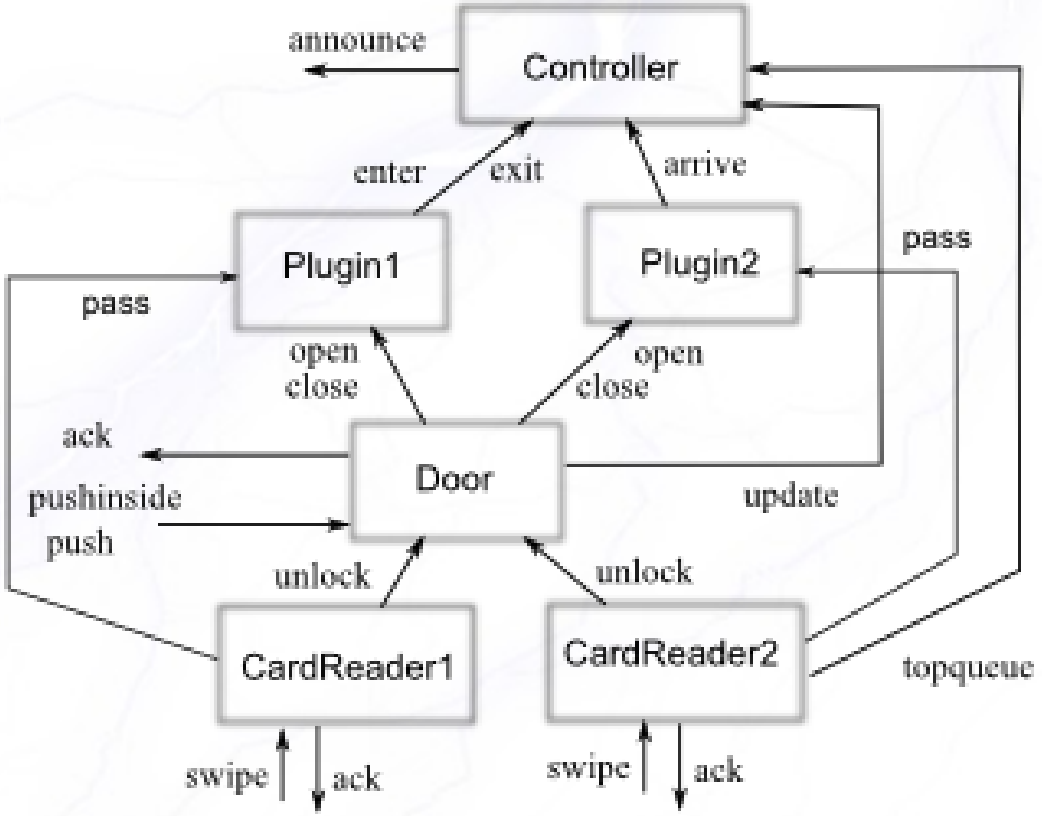
- The UserMachine generates events to activate the system.



- 'swipe', 'push' and 'push from inside' to the guiding system components.
- 'step' represents the user movement. It is wrapped and sent to the PolicyEnforcer

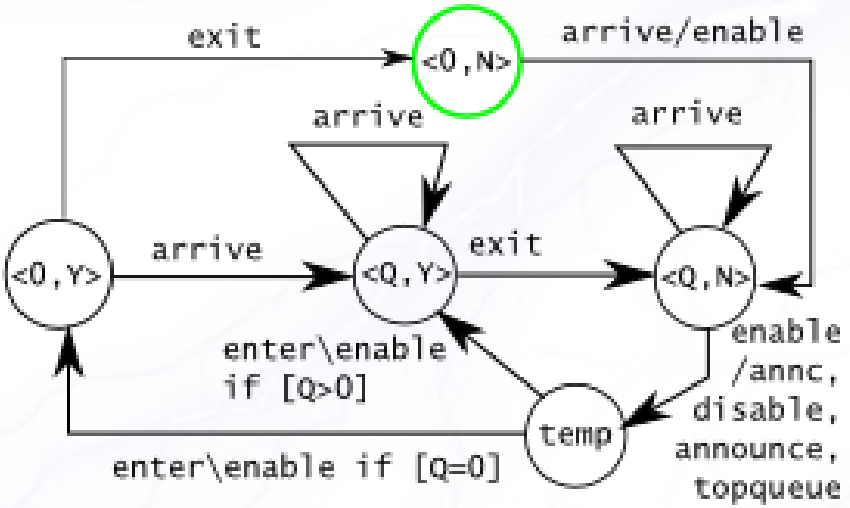
State Transition Diagram of Guiding System Machines

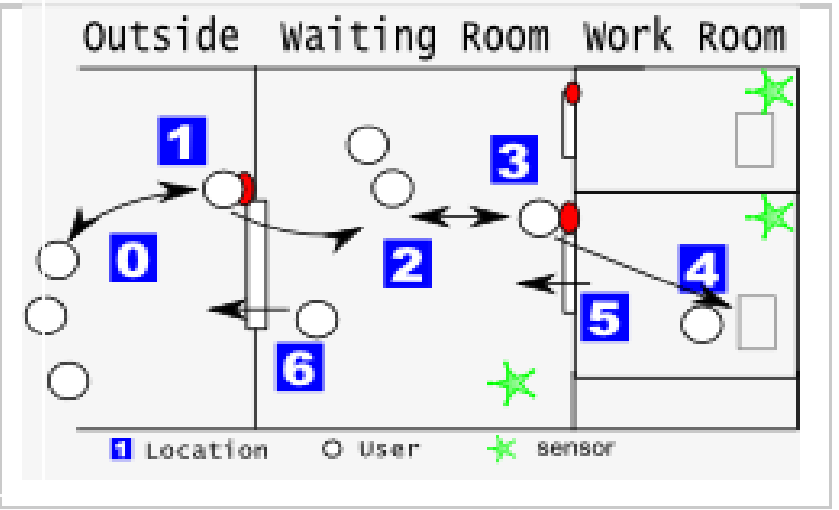
- The Guiding System machines response to UserMachine events.



Guiding System Controller

- Controller keeps a queue and a tag indicating the availability of the work room.





Enforce the Policy

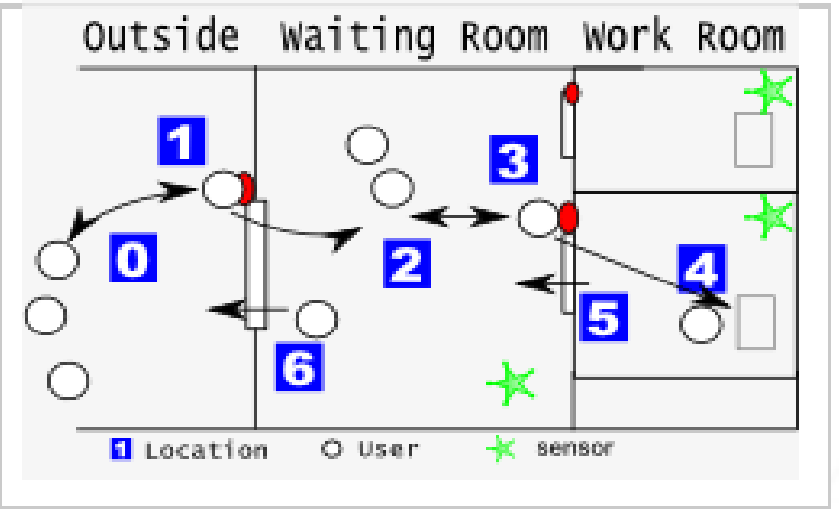
- We send the interesting UserMachine event to the PolicyEnforcer.
- PolicyEnforcer monitors the status of the system and decide if a UserMachine event is appropriate.
- The Policy Enforcer sends the 'good' events to the system, and corrects the 'bad' ones.
- Constantly moving between two locations can cause a live lock. If UserMachine generates such events, Policy Enforcer will 'stop' it.

Guiding System in Maude

- Summary
  - Location Graph
  - Breakdown the Guiding System
  - Extended Mealy Machine
  - The User Machine
  - Guiding System
  - Enforce the Policy
  - **Guiding System and Policy**

Guiding System and Policy

- UserMachine generates unusual behavior
- System is blocked.
- PolicyEnforcer detects the problem
- PolicyEnforcer resolves the problem
  - Notify system manager
  - Remove the User
    from the queue

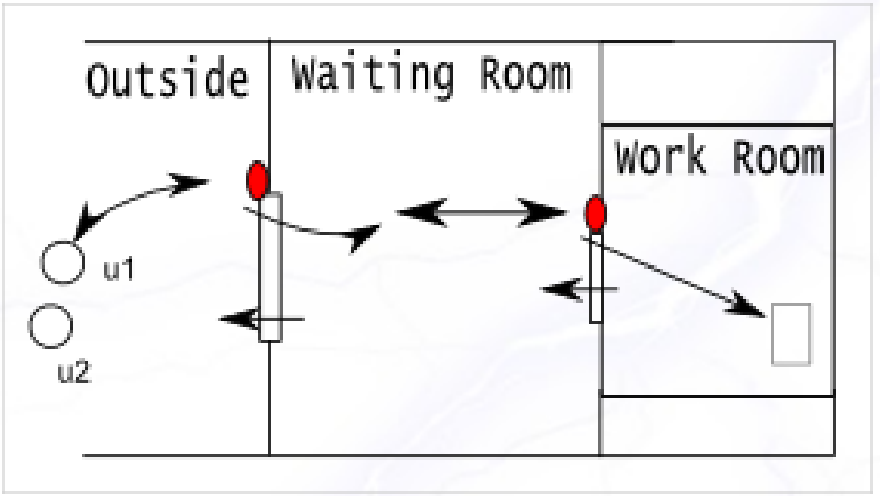


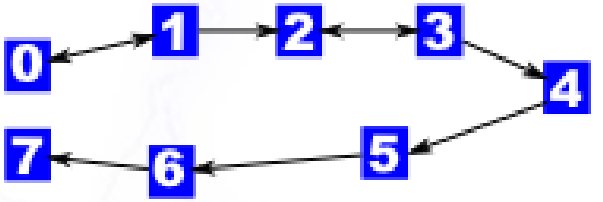
Content

- Background
- Issue and Our Approach
- Policy Enforcer
- Maude
- A Brief Introduction to the Guiding System
- Describe the Guiding System in Maude
- **Test and Result**
- Future Work

Test and Result

- Setup:





Test27 =
makeUser('u2) makeUser('u1) makeUser('u3) makeUser('u7)
makeUser('u4) makeUser('u6) makeUser('u5) m(inc(inc(inc(inc(inc
(inc(inc(nobody, L('L0)), L('L0)), L('L0)), L('L0)), L('L0)), L('L0)), L('L0)))
makeDoor('L1) makeCardReaderW1('L1, ('u1 'u2 'u3 'u4 'u5 'u6 'u7))
makeDoor('L3) makeCardReaderW2('L3)
makePlugin1('Plugin1, 'Gsys, 'L1)
makePlugin2('Plugin2, 'Gsys, 'L3)
makeGSys('Gsys) tog(false)
makePolicy('p) H* .

Test and Result

- Use the rewrite command. The output is like this

```
Maude> rew test27 .
rewrite in test : test27 .
rewrites: 3719 in 1628036047000ms cpu (187ms real) (0 rewrites/second)
result [Soup]: m(L('L7) |-> 7) tog(false) H* & 'u1 @ 'L1 & 'u1 @ 'L2 & 'u1 @
    'L3 & 'u1 @ 'L4 & 'u1 @ 'L5 & 'u1 @ 'L6 & 'u1 @ 'L7 & 'u2 @ 'L1 & 'u2 @ 'L2
    & 'u2 @ 'L3 & 'u2 @ 'L4 & 'u2 @ 'L5 & 'u2 @ 'L6 & 'u2 @ 'L7 & 'u3 @ 'L1 &
    'u3 @ 'L2 & 'u3 @ 'L3 & 'u3 @ 'L4 & 'u3 @ 'L5 & 'u3 @ 'L6 & 'u3 @ 'L7 & 'u4
    @ 'L1 & 'u4 @ 'L2 & 'u4 @ 'L3 & 'u4 @ 'L4 & 'u4 @ 'L5 & 'u4 @ 'L6 & 'u4 @
    'L7 & 'u5 @ 'L1 & 'u5 @ 'L2 & 'u5 @ 'L3 & 'u5 @ 'L4 & 'u5 @ 'L5 & 'u5 @ 'L6
    & 'u5 @ 'L7 & 'u6 @ 'L1 & 'u6 @ 'L2 & 'u6 @ 'L3 & 'u6 @ 'L4 & 'u6 @ 'L5 &
    'u6 @ 'L6 & 'u6 @ 'L7 & 'u7 @ 'L1 & 'u7 @ 'L2 & 'u7 @ 'L3 & 'u7 @ 'L4 & 'u7
    @ 'L5 & 'u7 @ 'L6 & 'u7 @ 'L7 < 'Gsys : 'GuideSys | 'O_N ; 'isbusy = 0,
    'qlen = 0,'queue = nil > < 'L1 : 'CardReaderW1 | 'idle ; 'db = ('u1 'u2 'u3
    'u4 'u5 'u6 'u7) > < 'L1 : 'Door1 | 'lock ; null > < 'L3 : 'CardReaderW2 |
    'idle ; 'user = ' > < 'L3 : 'Door1 | 'lock ; null > < 'Plugin1 : 'Plugin1 |
    'idle ; 'door = 'L1,'gsys = 'Gsys,'usr = ' > < 'Plugin2 : 'Plugin2 | 'idle
    ; 'door = 'L3,'gsys = 'Gsys,'usr = ' > < 'p : 'Policy | 's0 ; null > < 'u1
    : 'User | L('L7) ; 'atPosition ; null > < 'u2 : 'User | L('L7) ;
    'atPosition ; null > < 'u3 : 'User | L('L7) ; 'atPosition ; null > < 'u4 :
    'User | L('L7) ; 'atPosition ; null > < 'u5 : 'User | L('L7) ; 'atPosition
    ; null > < 'u6 : 'User | L('L7) ; 'atPosition ; null > < 'u7 : 'User | L(
    'L7) ; 'atPosition ; null >
Maude>
```

Future Work

- Investigate the policy for different application.
- Develop the current example to simulate more complex, more real situations.
- Make use more Maude analysis functionality.

Thanks & Any Questions? ^_^