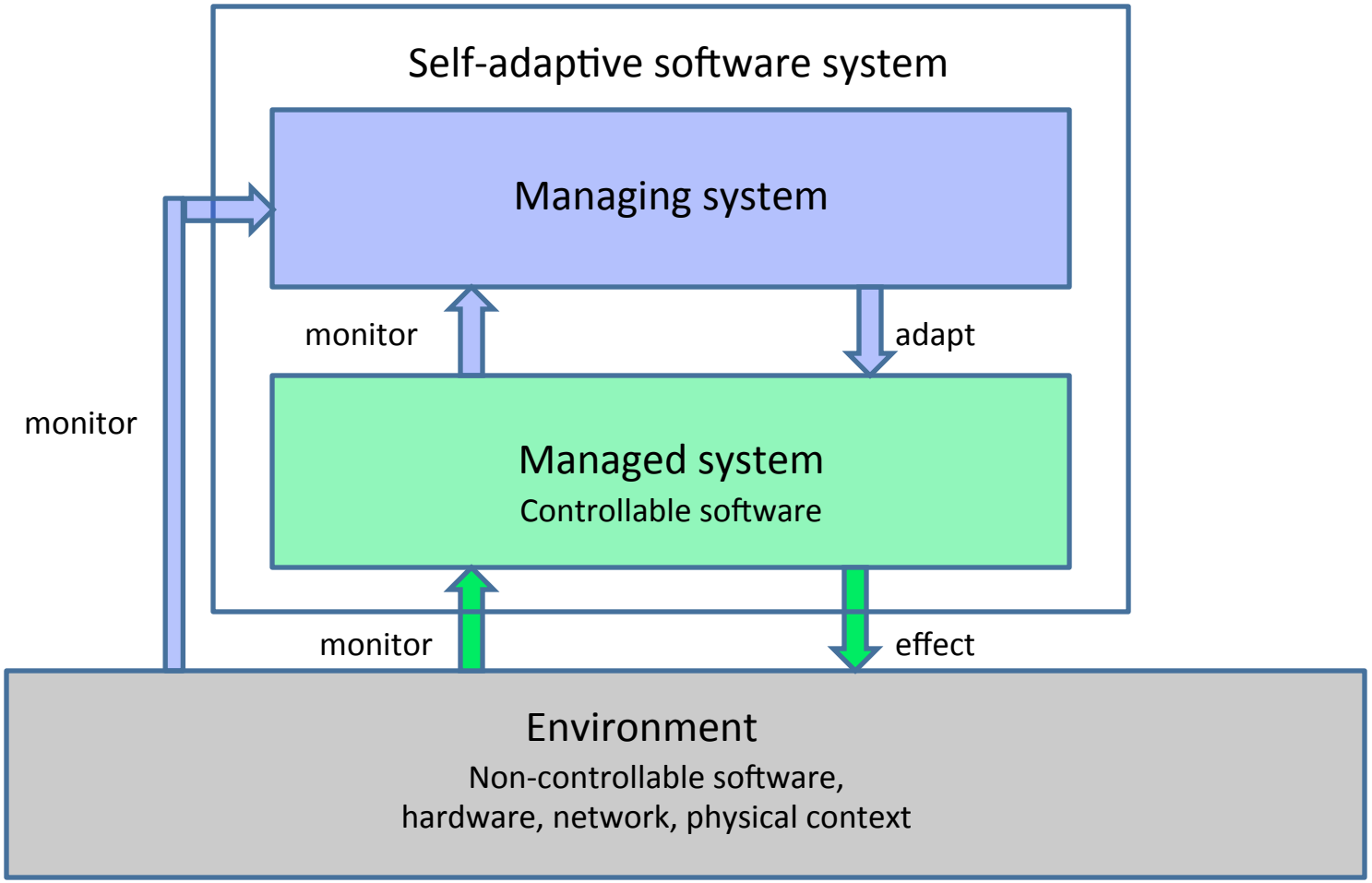
ActivFORMS: Active formal models for Self-Adaptation

M. Usman Iftikhar, Danny Weyns
Linnaeus University Sweden
{usman.iftikhar, danny.weyns}@lnu.se
http://homepage.lnu.se/staff/daweaa/ActivFORMS/ActivFORMS.htm

Basic model architecture-based self-adaptation



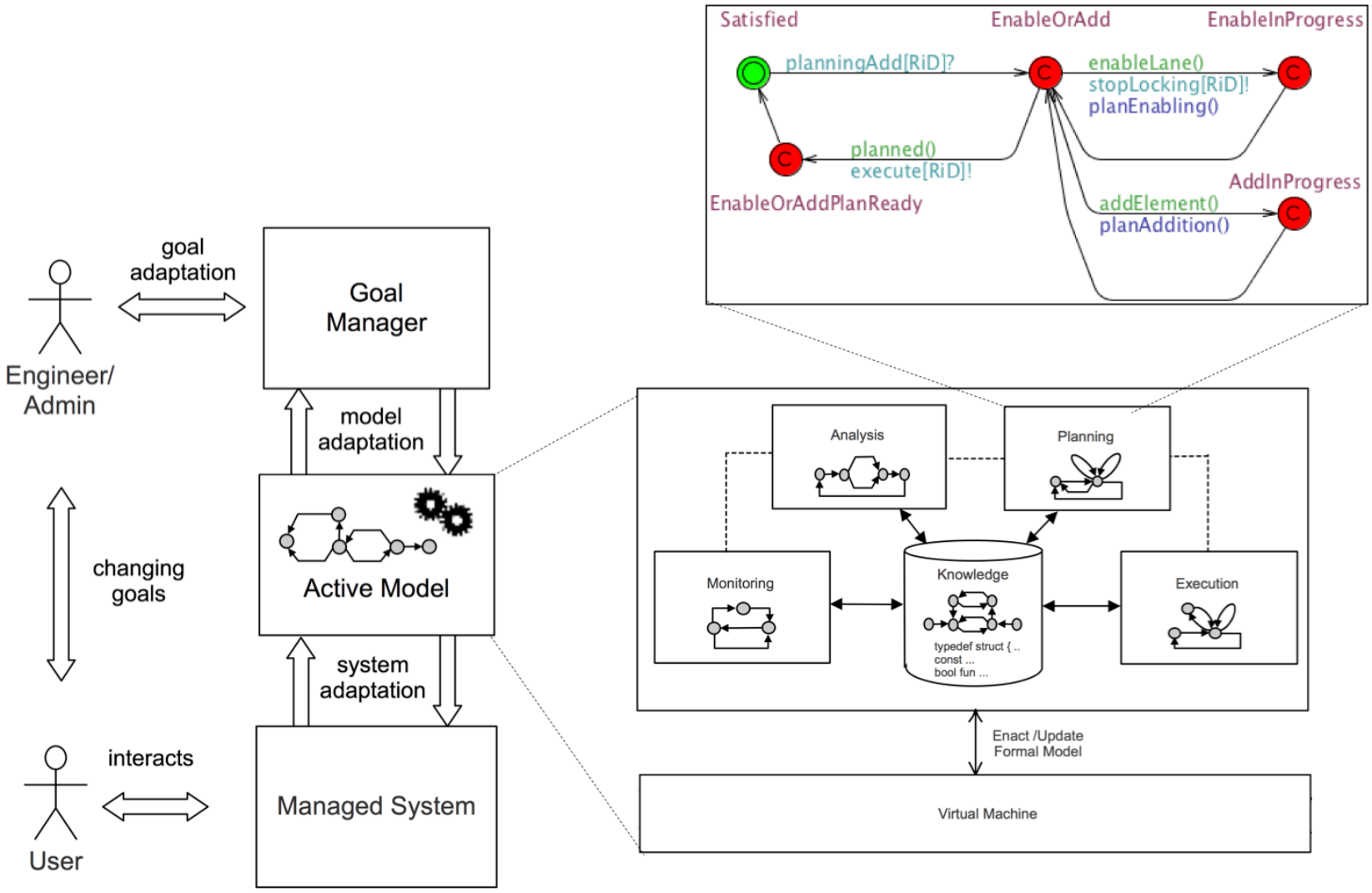
ActivFORMS

Active formal models for self-adaptation

- Formalization of complete MAPE-K loop
- Model is directly executed to adapt the managed system
- Model directly supports online verification of goal satisfaction/violation
- Model can be adapted at runtime to support unanticipated changes

http://homepage.lnu.se/staff/daweaa/ActivFORMS/ActivFORMS.htm

Approach

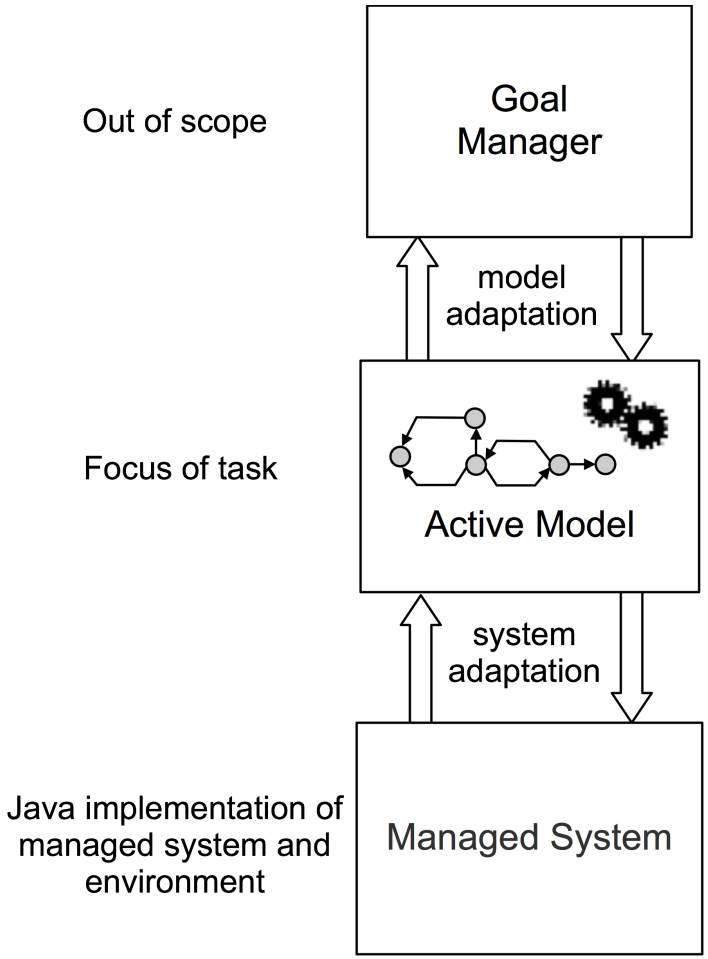

Satisfied
EnableOrAdd
EnableInProgress
planningAdd[RiD]?
enableLane()
stopLocking[RiD]!
planEnabling()
planned()
execute[RiD]!
EnableOrAddPlanReady
AddInProgress
addElement()
planAddition()
goal adaptation
Goal Manager
Engineer/ Admin
model adaptation
changing goals
Active Model
system adaptation
interacts
Managed System
User
Analysis
Planning
Monitoring
Knowledge
typedef struct { ..
const ...
bool fun ...
Execution
Enact /Update Formal Model
Virtual Machine

Levels of adaptation

- Level 1: active model adapts the managed system

- Level 2: adapt the active model (adapt MAPE)

Task


Out of scope
Goal Manager
model adaptation
Focus of task
Active Model
system adaptation
Java implementation of managed system and environment
Managed System

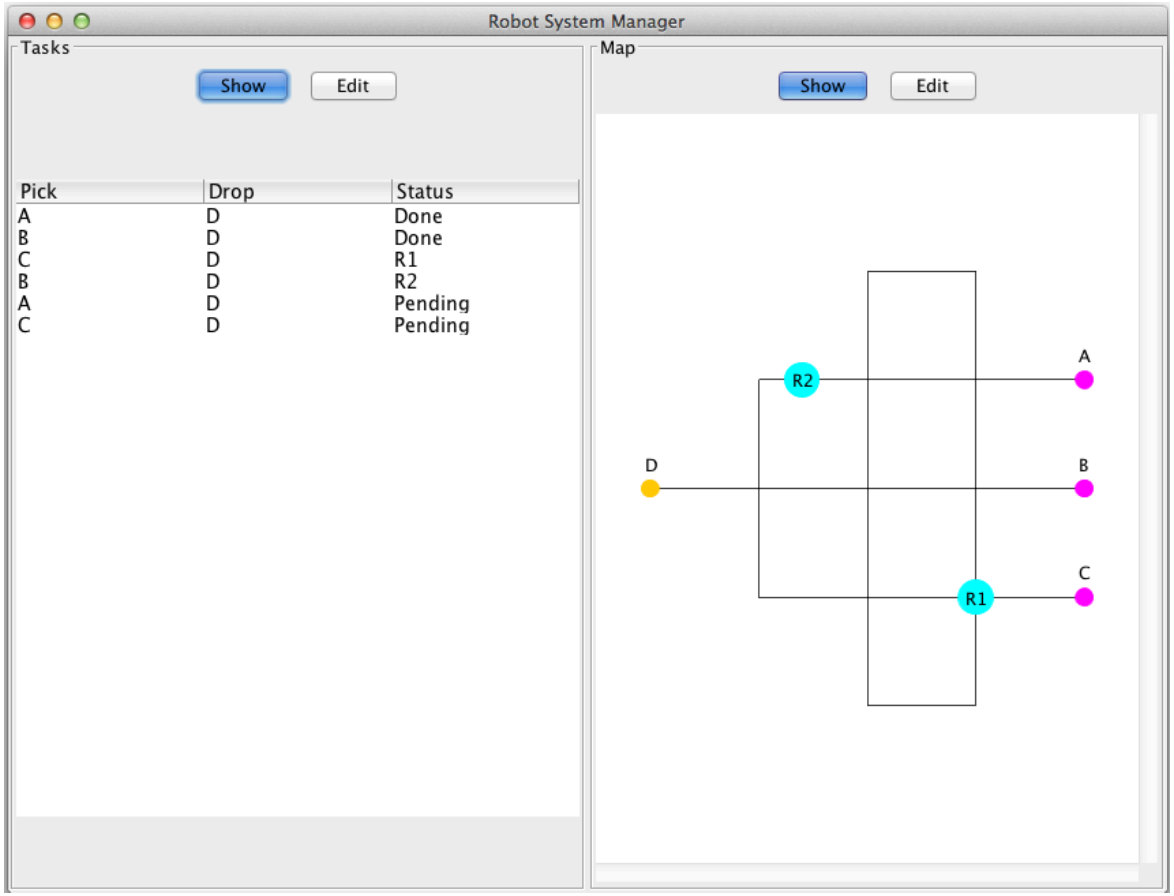
Goal Manager

- Provides interface for runtime management of
  - Runtime verification of system goals
  - Runtime adaptation of the managing system

Virtual machine

- Transforms a formal model (network of timed automata) into a task graph representation
- Executes that model
- Can detect and notify goal violations
- Can adapt the current model at runtime

Case study


Robot System Manager
Tasks
Show
Edit
Pick
Drop
Status
A
D
Done
B
D
Done
C
D
R1
B
D
R2
A
D
Pending
C
D
Pending
Map
Show
Edit
R2
A
D
B
R1
C

User Interface

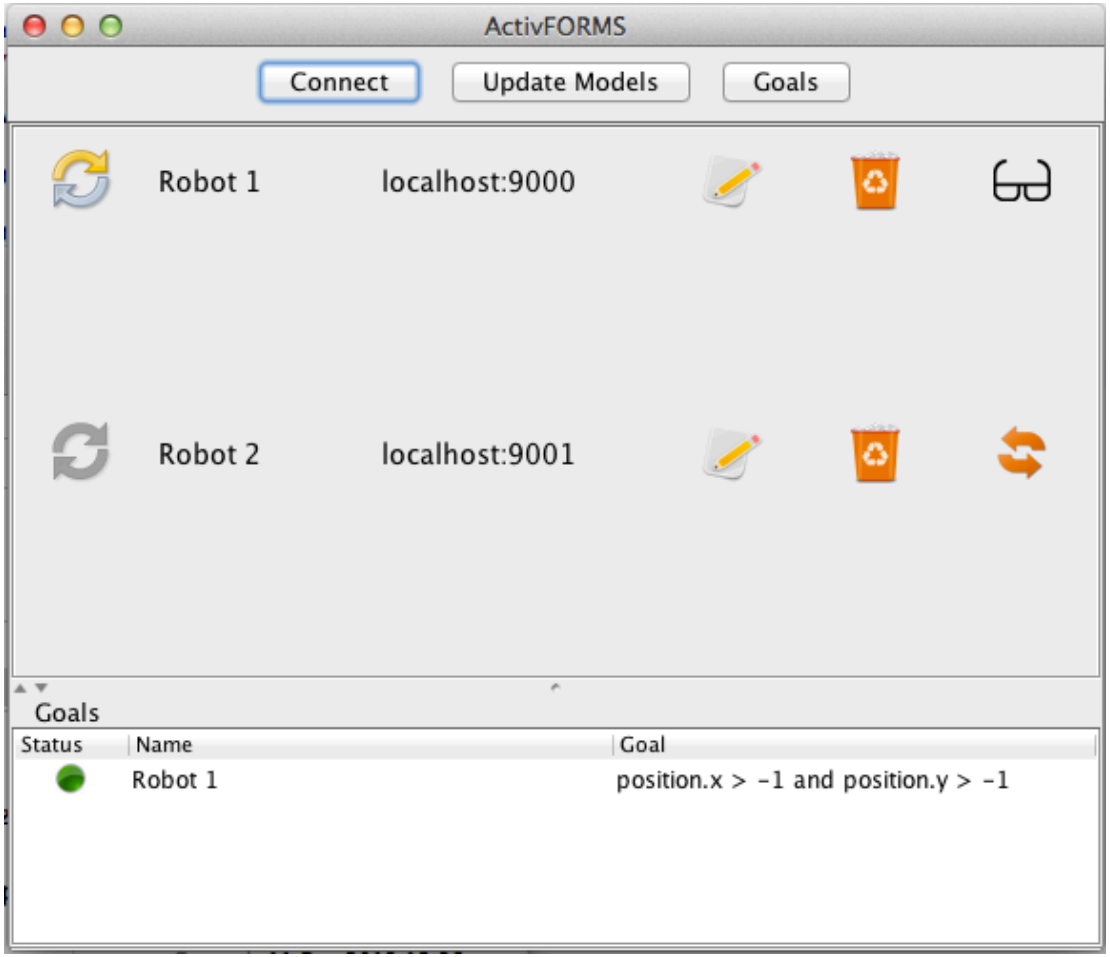
- Download ActivFORMS with the extra required libraries from:

  http://homepage.lnu.se/staff/daweaa/ActivFORMS/ActivFORMS.htm

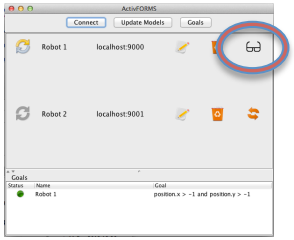
- To start ActivFORMS double click on the ActivFORMS jar file
- Or run activforms.gui.ActivFORMS via the command line:

  "java -jar ActivFORMS.jar activforms.gui.ActivFORMS"

User Interface



View Running Model



Robot 1

Globals | Analyze | Planning_Add | Planning_Rem | Executor_Lock

Globals

Expression	Value
destinations	{0={0={x=0, y...
occupied	false
remLocation	{0=channel}
pos_update	{0=0}
LockZ	101
LockY	100
addedLane	{0={src={x=0,...
jobRequest	channel
DIRECTIONS	4
SOUTH	2
posUpdate	{0=channel}
rem_loc	{0=0}
empty	{x=-1, y=-1}
ok	channel
addLocation	{0=channel}
updateDest	{x=0, y=0}
PLANNING	2
ROBOTS	1
waiting	channel
sDest	{x=0, y=0}
remDestination	{0=channel}
charging	50
cPosition	{0={x=0, y=0}}
update_position	channel
EDGE_UNAVAIL...	0
STARTING_SOU...	{0={x=4, y=1}...
PLANNED	3
STARTING_DES...	{0={x=0, y=2}...
remLane	{0=channel}
LockE	102
startLocking	{0=channel}
RECVD	1
position	{x=0, y=0}
D	{x=0, y=2}

Analyze

Analyzing
Extra == matchResources(request)
planningRem[RiD]!
DisableOrRemoveRequest
analyze[RiD]?
noNeed == matchResources(request)
NoChange
analyze[RiD]?
Missing == matchResources(request
planningAdd[RiD]!
EnableOrAddRequest
analyze[RiD]?
request = getRequest()

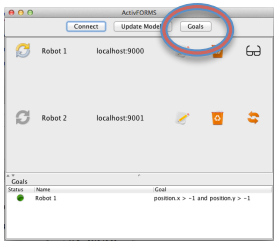
Planning_Add

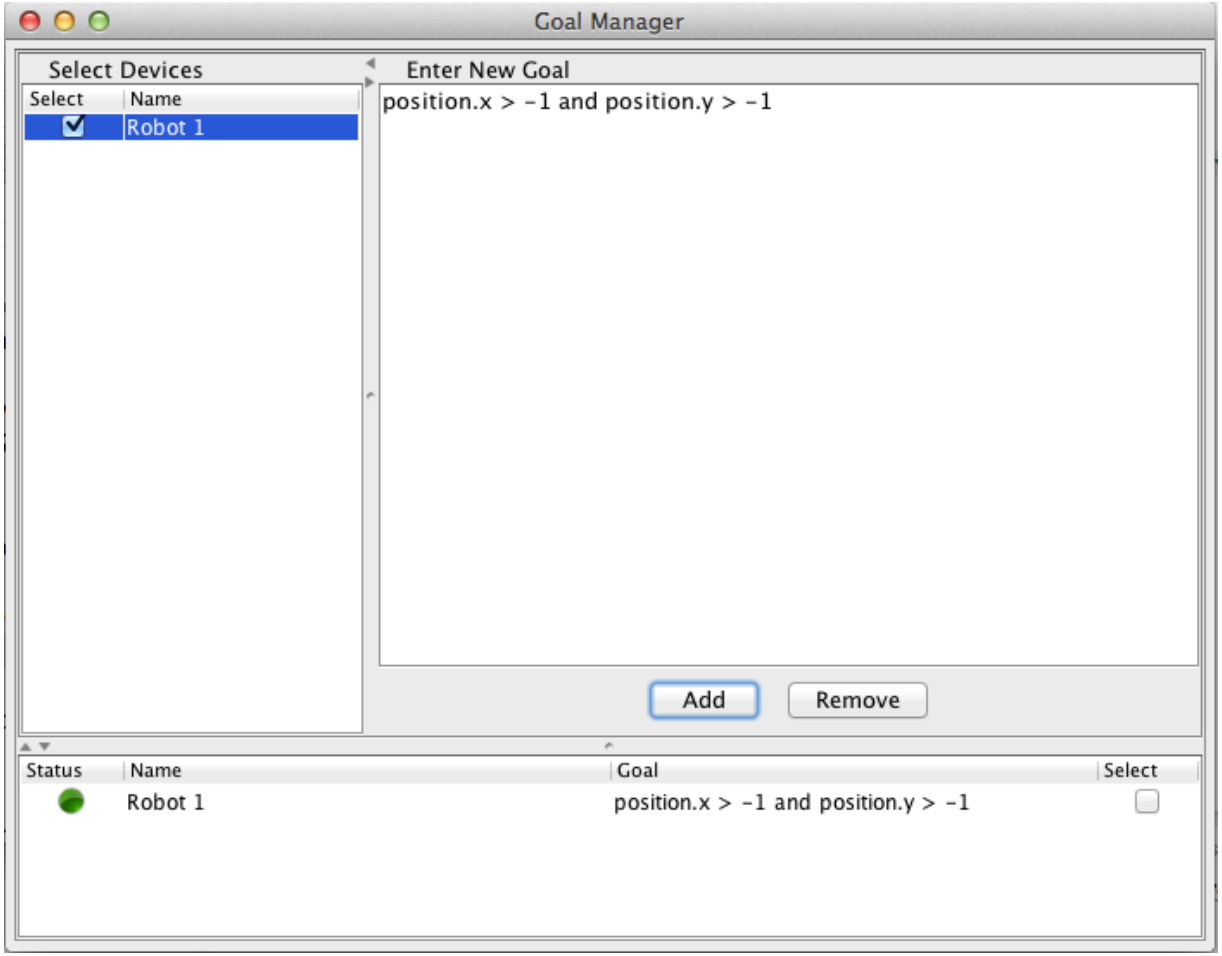
Satisfied
EnableOrAdd
EnableInProgress
planningAdd[RiD]?
enableLane()
planEnabling()
planned()
execute[RiD]!
EnableOrAddPlanReady
AddInProgress
addElement()
planAddition()

Planning_Rem

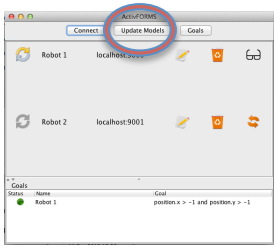
planningRem[RiD]?
DisableInProgress
DisableOrRemove
Satisfied

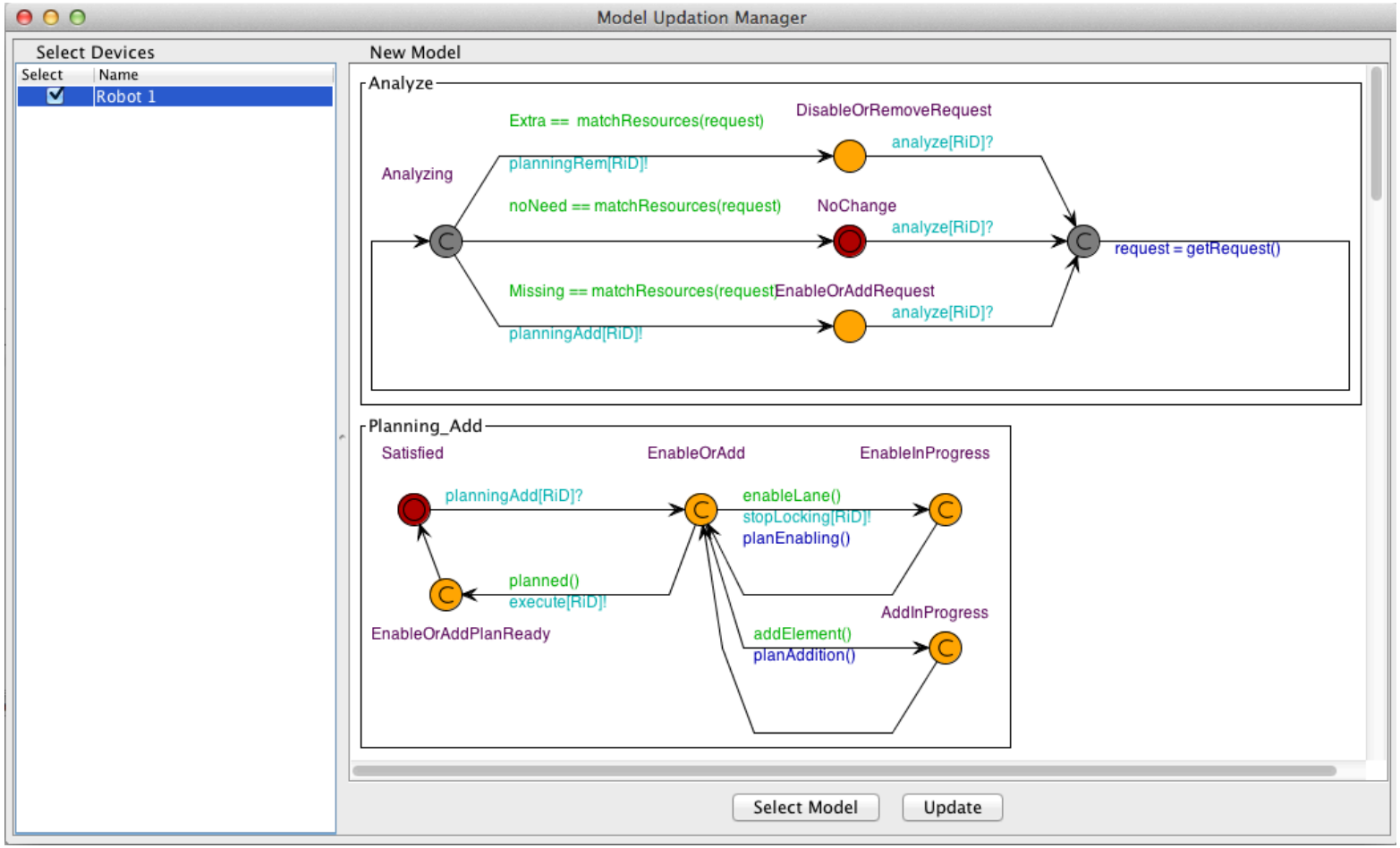
Goal Management





Model Updates





How to use ActiveFORMS?

1. Select models
2. Configure virtual machine
3. Create probes and effectors
4. Send and receive data
5. Start virtual machine

1. Select Models

Select the managing processes that need to be executed by ActivFORMS.

e.g. system, monitor, analyze, plan, execute;

2. Configure Virtual Machine

- Add ActivFORMS.jar into your java project as external jar with required libraries provided in lib folder
- Import activforms.engine.ActivFORMSEngine;
- Instantiate Virtual Machine

  `ActivFORMSEngine` *engine = new* `ActivFORMSEngine`*(path, port);*
- Set maximum number of delay transition at a time

  engine.setMaximumNoOfDelayTransitions(1);
- Set model time unit to real time

  engine.setRealTimeUnit(1000);
- Set committed location time to have good view in gui if needed

  engine.setCommittedLocationTime(500);

3. Create probes and effectors

- Probes: send signals to the managing system from the managed system

- Effectors: receive signals from the managing system for the managed system

Probe

- Virtual machine uses unique identifier for each channel. E.g.

  int monitor = engine.getChannel("monitor");

- Use channel ids to send signal from managed system to managing system

  engine.send(monitor, synchronizer, data);

Effector

- An effector must register itself to the virtual machine for the channels from which it wants to receive data. E.g.

int effector = engine.getChannel(“effector[0]”);

engine.register(effector, synchronizer, data);

Synchronizer

- An interface that is used by the virtual machine to communicate with the managed system. You have to import activforms.engine.Synchronizer in your class to use it. Synchronizer has three functions:

- Return true if the managed system (via probes and effectors) is ready to receive a signal, however signal is not guaranteed.

  boolean readyToReceive(int channelId);

- The managed system receives the signal after response to *readyToReceive*.

  void receive(int channelId, HashMap<String, Object> data);

- When the managed system sends a signal, the virtual machine sends an acknowledgment

  void accepted(int channelId);

4. Send and Receive data

- Probe sends data to virtual machine by calling
  engine.send(channelId, synchronizer, data);

- Data are a number of string expressions
  - *“request=3”,*
  - *“array[0]=2”,*
  - *“struct.member.value=3”*
  - *“struct.array[index] = 4”*

4. Send and Receive data

- Effector registers for signals at the virtual machine
  engine.register(channelId, synchronizer, data);

- Data are a number of string expressions to receive with each signal
  - *“request”,*
  - *“array[0]”,*
  - *“struct.member.value”*
  - *“struct.array[index]”*
  - *“struct”*

- Data will be sent with each call to receive function of synchronizer

Example Probe

```
public class MyProbe implements Synchronizer {
        int monitor;
        public MyProbe (ActivFORMSEngine engine){
                monitor = engine.getChannel("monitor");
        }

        public sendSignal(){
                engine.send(monitor, this, dataToSend);
        }

        @override
        public accepted(int channelId){
                if (monitor == channelId)
                        system.out.println("Monitor signal is accepted");
        }

        /// other functions
        ……
}
```

Example Effector

```
public class MyEffector implements Synchronizer {
        int effector;
        public MyEffector (ActivFORMSEngine engine){
                effctor= engine.getChannel("effector");
                effector.register(effector, this, dataNeeded);
        }

        @override
        public readyToReceive(int channelId){
                if (channelId == effector)
                        return true;
        }
        @override
        public receive(int channelId, HashMap<String, Object> data){
                if (monitor == effector){
                        system.out.println("Data:" + data);
                }
        }

        /// other functions
        …….
}
```

5. Start virtual machine

- Call "*start*" method to start the virtual machine, i.e.

  engine.start();

- If the virtual machine blocks it will throw a runtime exception

Features not supported

- Scalar types and meta variables
- Priorities of channels and processes
- Selection annotation of edges
- Iteration loop
- Forall, exit and sum functions
- Invariants are not shown in the GUI

Summary

- Formal active model guarantees verified properties of the adaption process
- Active model directly executes the adaptation: no coding, no model transformations
- Adaptation of adaptation functions: lightweight process to add new goals
- Online detection of goal violations