unibz

SOFTWARE AND SYSTEMS ENGINEERING research group

# Model-based testing under uncertainty

Hands-on session

Matteo Camilli
matteo.camilli@unibz.it
https://matteocamilli.github.io

[slides] https://github.com/matteocamilli/mbt-uncertainty-gssi

**Formal Methods at Work @**
**Gran Sasso Science Institute (GSSI)**
**A.Y. 2021/22**

G S
S I

# Outline

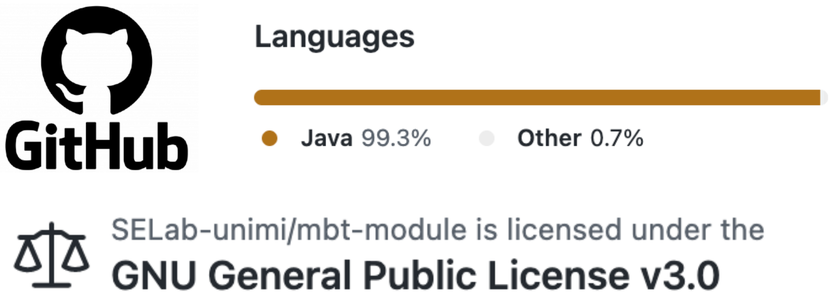
- How do I get setup?
  - The MBT module
  - Virtual machine (Ubuntu 20)
  - Requirements + Quick installation guide
  - Try out the VM
- First steps
  - A running example
  - Inspect and then run
  - Try out different testing strategies
- Advanced exercise
  - Develop a new testing strategy

# How do I get setup?

- The MBT module
- Virtual machine (Ubuntu 20)
- Requirements + Quick installation guide
- Try out the VM

# The MBT module

- The project
  - Open source software
  - https://github.com/SELab-unimi/mbt-module
  - Java but uses R for statistical machinery
  - implementation of MBT testing strategies[1]
    - Random / Flat
    - History
    - Distance
    - Frequency




1. M. Camilli, A. Gargantini, P. Scandurra and C. Trubiani, "Uncertainty-aware Exploration in Model-based Testing," 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), 2021, pp. 71-81, doi: 10.1109/ICST49551.2021.00019

# Virtual machine

- Requirements
  - Virtual Box — https://www.virtualbox.org/wiki/Downloads
  - 20 GB free space on disk
- Setup
  - Download the OVA image
    - https://drive.google.com/file/d/19WMFR-UjZ7YjY8REw2gfuZlJwJjO6Xg0/view?usp=sharing
  - Import the image into VB using “File > Import Appliance…”
  - Start the new VM “ubuntu64-mbt-machine”

# Requirements

- Operating system
  - Linux / macOS
- Environment
  - JDK version 1.8 (Oracle / OpenJDK) — https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
    - Set the JVM 1.8 as default one
  - R version >= 4.0 — https://www.r-project.org/
    - R packages
      - rJava — https://cran.r-project.org/web/packages/rJava/index.html
      - gtools — https://cran.r-project.org/web/packages/gtools/index.html
      - HDInterval — https://cran.r-project.org/web/packages/HDInterval/index.html

# Installation

- MBT-module
  - Open source Git repository — https://github.com/SELab-unimi/mbt-module
  - Clone the repository
  - Follow the README

# Build + execution

- From the mbt-module root folder by using Gradle build tool

```
> ./gradlew clean build
…
BUILD SUCCESSFUL in …s

> ./gradlew run -PappArgs="['-i', ‘src/main/resources/roboticarm.jmdp']"

Value Iter. Solver (Avg)
… states found.
********* Best Policy *********
…
Log trace
…
#test limit reached.
********* Monitor report *********
…
```

# Import into the IDE

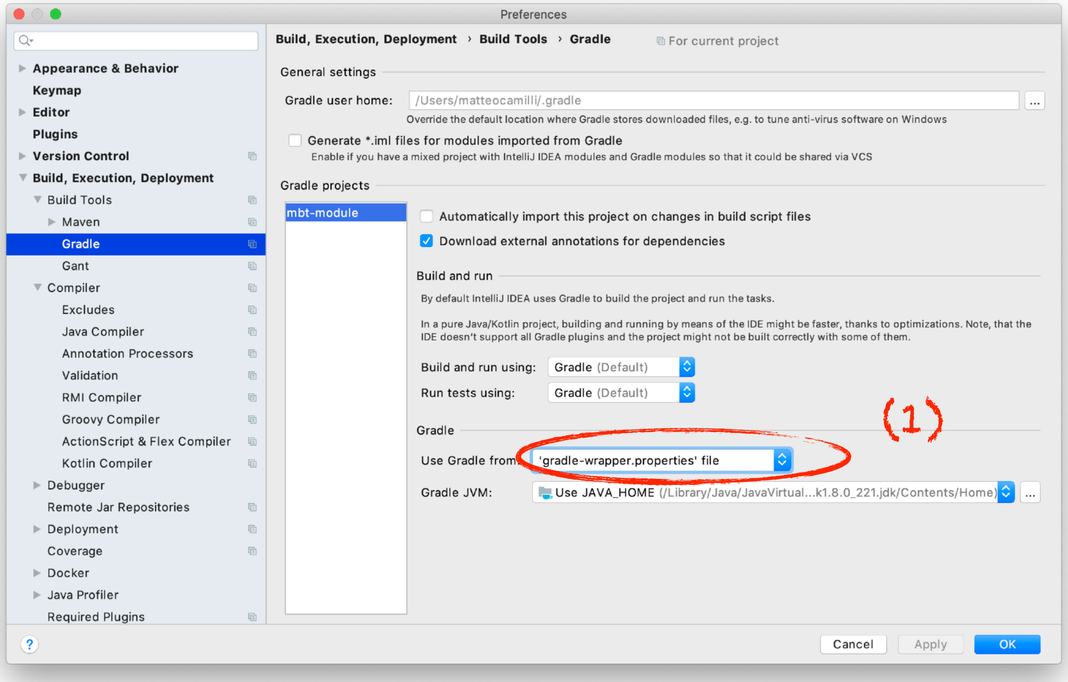
- Need: aspectJ and grade integration

- Intellij Idea IDE
  - https://www.jetbrains.com/idea/
  - Ultimate edition needed (it includes the aspectJ plugin)
- Eclipse IDE
  - https://www.eclipse.org/downloads/
  - AspectJ Development Tools (AJDT) plugin
    - https://marketplace.eclipse.org/content/aspectj-development-tools

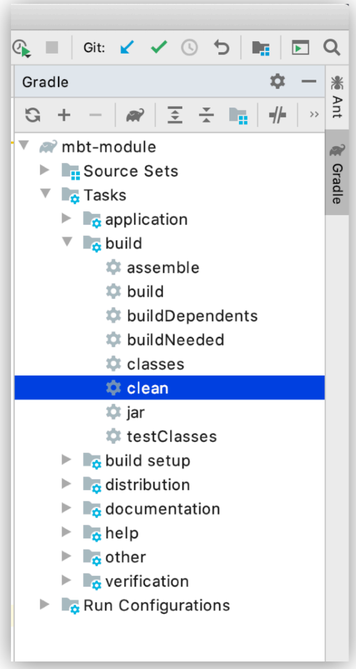
- Open/import an existing gradle project
- Select the mbt-module directory

# Build from the IDE

- Clean + build
  - Be sure that **gradle-wrapper** option is selected
  - Use the IDE to run **gradle** commands: **clean** and then **build**

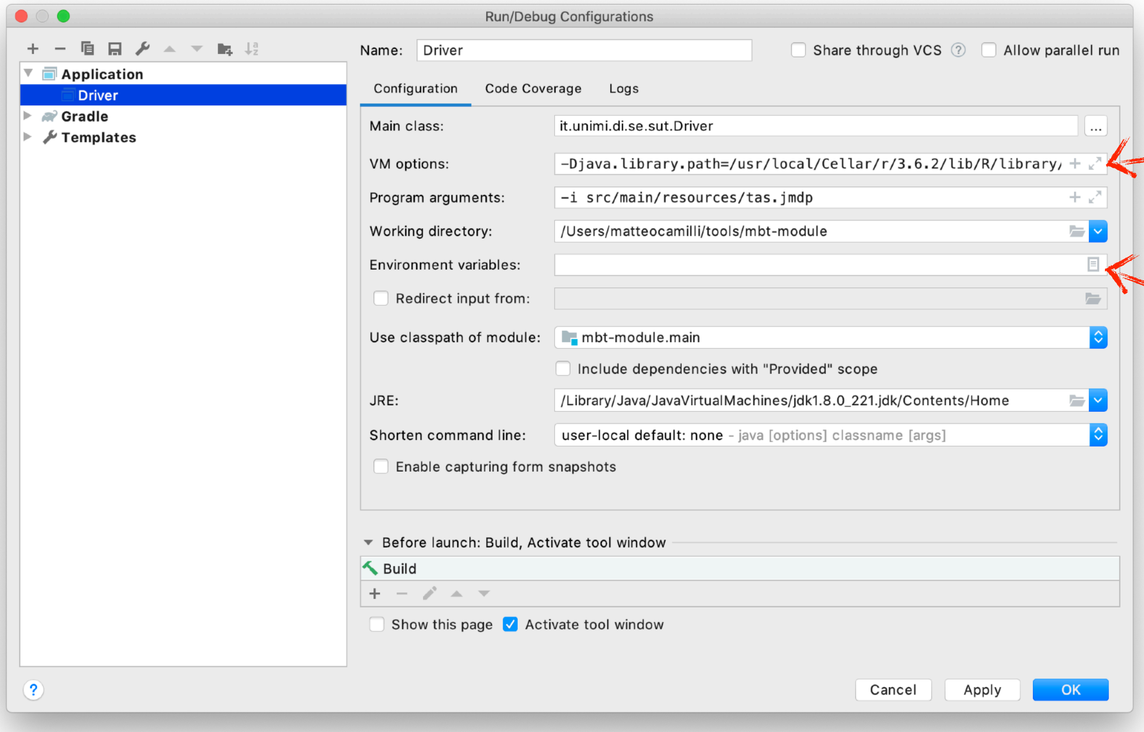



(2)



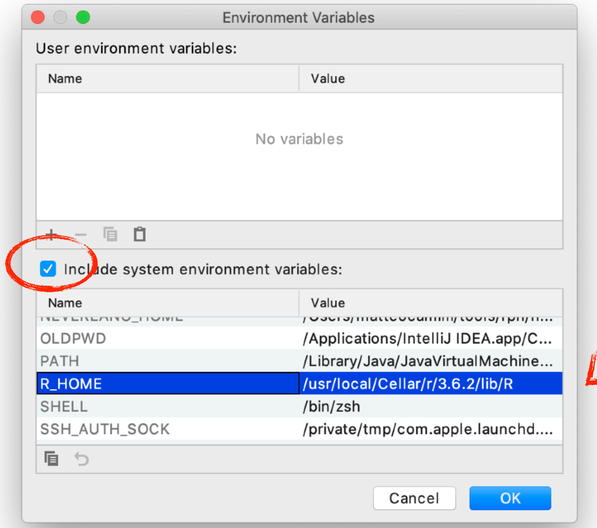
# Run from the IDE

- Run configuration
  - Execute the Driver class with the following configuration




VM options (taken from build.gradle)

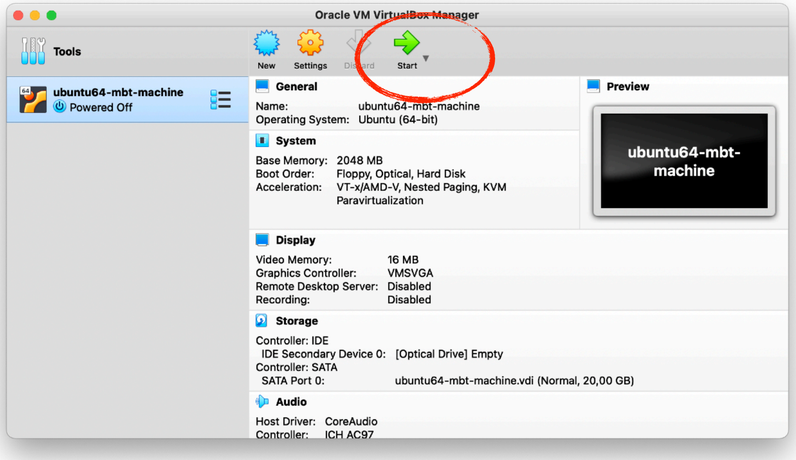
-Djava.library.path=/usr/local/Cellar/r/3.6.2/lib/R/library/rJava/jri
-Dorg.slf4j.simpleLogger.defaultLogLevel=warn



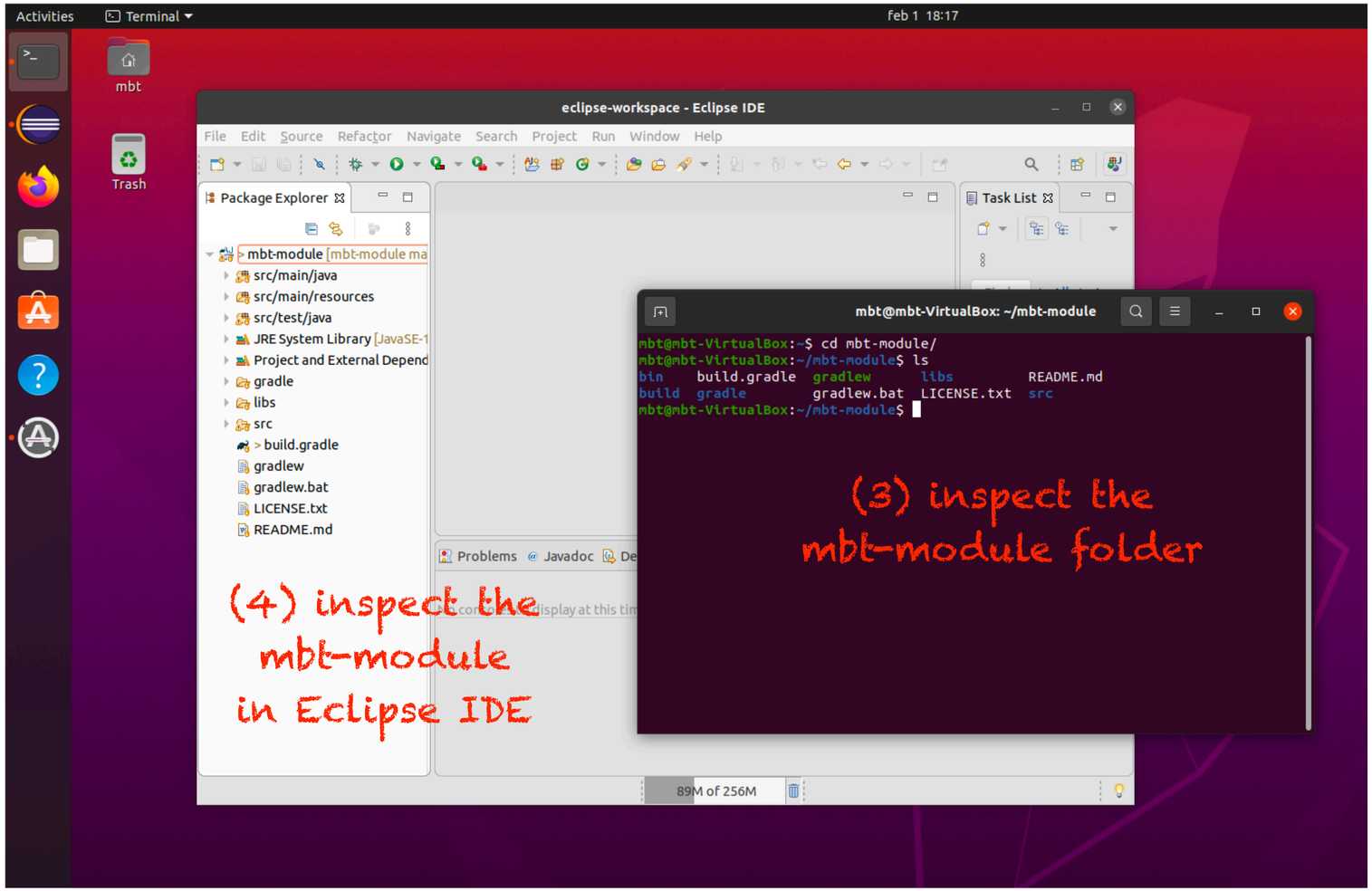

R_HOME must be defined

# Try out the VM

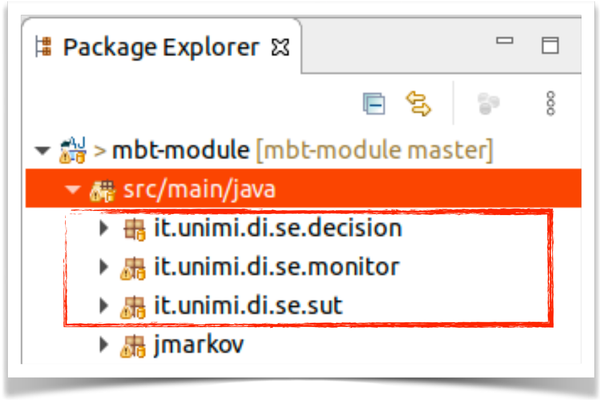
(1) Start the VM



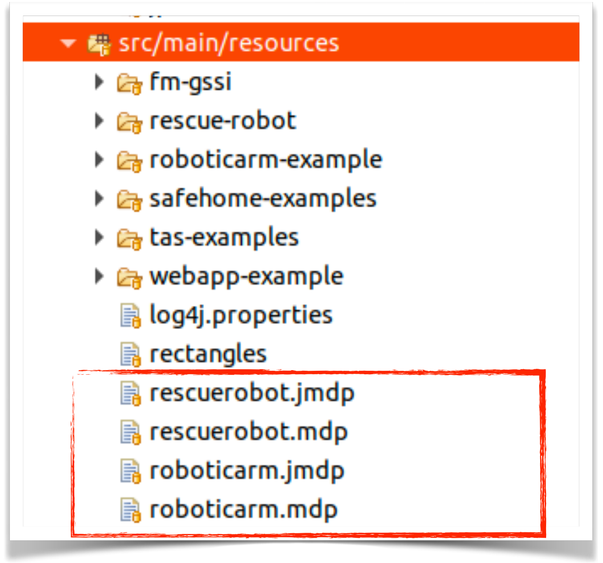

(2) login: mbt2022



# Project structure




Relevant packages in src/
- decision: implementation of the strategies
- monitor: instrumentation (test harness)
- SUT: system under test (simulator)



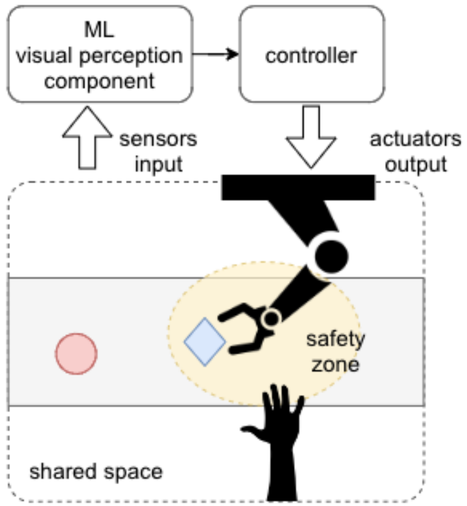

Relevant files in resources/
- *.jmdp: definition of the system to be simulated
- *.mdp: definition of the binding (model <-> SUT)

# First steps

- A running example
- Inspect and then run
- Try out different testing strategies

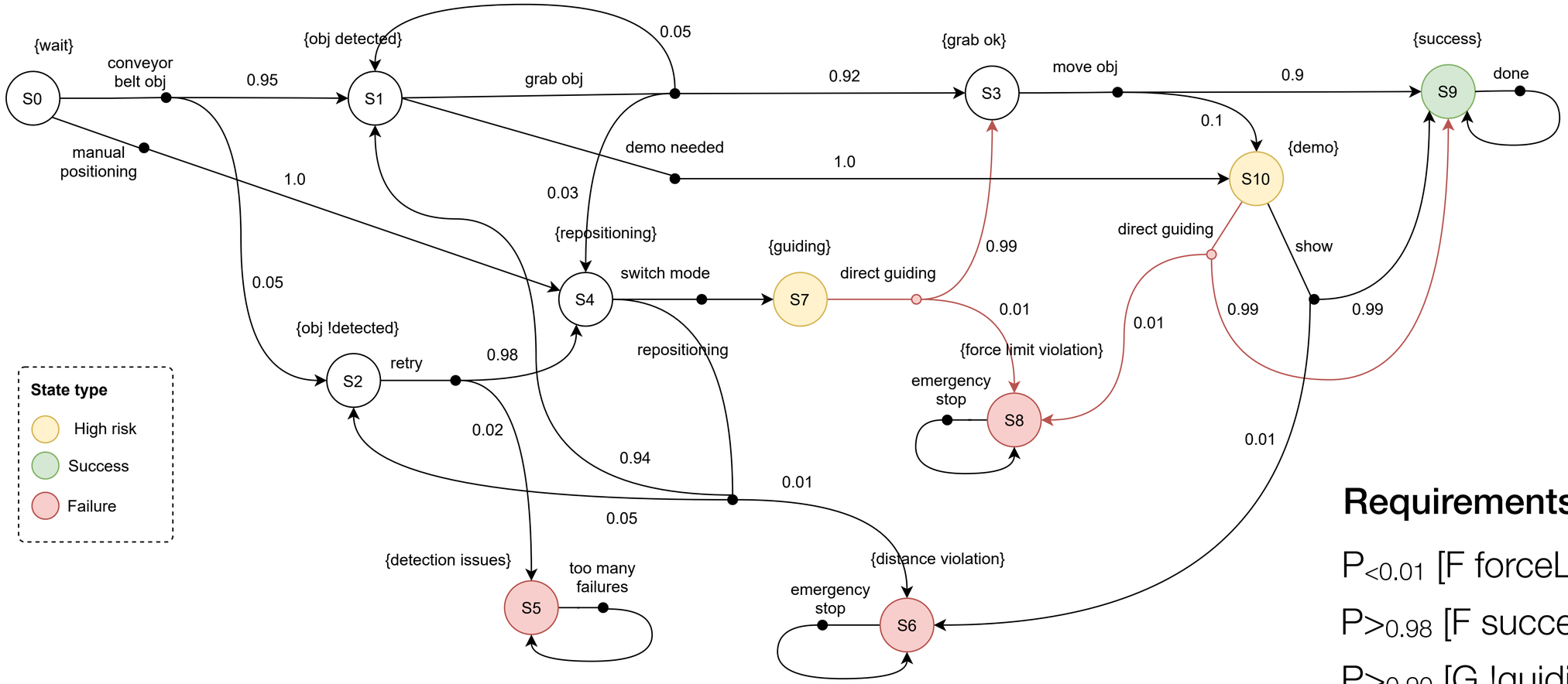
# A running example: robotic arm

- Industry 4.0 demo running at the ARENA Lab[1] (Fraunhofer Italia)
- Robotic arm detects and moves object from a conveyor belt into the right bucket
- The system is trained online by a human operator that can either:
  - show how to handle objects (protective separation shall be maintained); or
  - guide directly the movements of the arm (power and force limiting shall be applied)


1. https://www.fraunhofer.it/de/leistungsangebot/fraunhofer-italia-application-center-arena-.html

# Example — human-robot collaboration (MDP)




## Requirements (PCTL)

$P_{<0.01}$ [F forceLimitViolation]

$P_{>0.98}$ [F success]

$P_{>0.90}$ [G !guiding & !demo]

# (1) Inspect and run the robotic-arm example

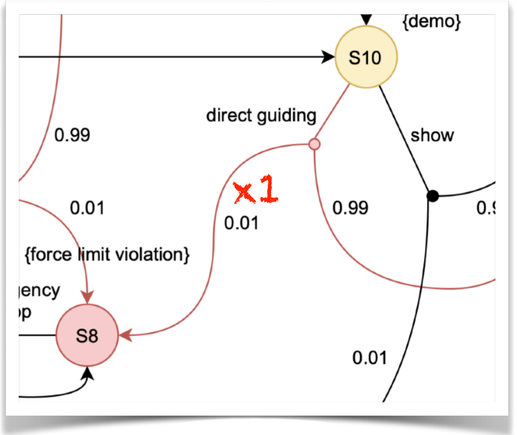
- Robotic-arm example
  - All the necessary files are already available in src/main/resources/
    - [SUT] simulated system — src/main/resources/roboticarm.jmdp
    - [MDP] model of the system — src/main/resources/roboticarm.mdp
    - [test harness] src/main/java/…/monitor/EventHandler.aj

- Inspect the [MDP]
  - Can you recognize the uncertain regions defined in this file?
  - What are the concentration parameters used to define the Dirichlet priors?
- Inspect the [test harness]
  - What strategy is used to initialize the Monitor object? What is the termination condition?
- Run the testing process
  - Run the testing process by using Gradle from the command line
  - What is the outcome you get for each uncertain region?

# (2) Try out different testing strategies

- Context
  - Suppose we have a safety requirement R1 that sets the following constraint for the “demo” state (S10)
    - R1: The “direct guiding” action shall lead to a “force limit violation” with probability less then 0.015

- Compare the effectiveness of FLAT vs RANDOM
  - Use `Policy.RANDOM` 10 times and for each run:
    - See the HDR calculated for the uncertain region (S10, e1)
    - Count how many times the testing detects violations of R1, i.e., the bounds for x1 exclude values less than 0.015
  - Repeat this experiment with `Policy.UNCERTAINTY_FLAT`
  - Which strategy is likely to be superior to detect violations?



# (3) Try out different testing strategies

- Compare the accuracy of FLAT vs DIST
  - Execute `Policy.UNCERTAINTY_FLAT` 10 times
  - Execute `Policy.DIST` 10 times
  - For each run, measure the difference of the two HDR size (for the two regions)
  - Which strategy is likely to reduce such a difference?

# Advanced exercise

- Design/implement a new testing strategy

# Implement an adaptive strategy

- Problem
  - The best policy for each region is computed at the beginning (based on prior knowledge)
  - This may lead to suboptimal exploration in case the initial assumptions are wrong
- Idea: implement an “adaptive” strategy
  - Every #sample-size tests, the policy for each regions shall be updated based on the posterior knowledge
  - This means that every the policies must be recalculated after updating the transition probabilities of the MDP model

# (3) Implement an adaptive strategy — hints

- Hints
  - The testing strategies follow the strategy design pattern and extend the DecisionMaker class
  - The DecisionMaker interface should be extended with a new method updatePolicy called by the Monitor every sample-size observations

Monitor.java

```
326             if(tests % EventHandler.SAMPLE_SIZE >= EventHandler.SAMPLE_SIZE-1) {
327                 //log.info(coverageInfo.toString());
328                 if(EventHandler.TERMINATION_CONDITION == Termination.COVERAGE && coverageInfo.getCo
329                     log.info("[Monitor] convergence reached.");
330                     addEvent(Event.stopEvent());
331                 }
```

else call updatePolicy

  - The new method updatePolicy should take as input the posteriors to update the MDP model and then recalculate the best exploration policies

# License of these slides