unibz

SOFTWARE AND SYSTEMS ENGINEERING research group

Model-based testing under uncertainty

L2: 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. 2020/21

G S S I

Outline

- How do I get setup?
  - By using Gradle from command line
  - By using your IDE of choice
- Icebreaker exercise
  - Try out pre-defined examples
- Advanced exercise
  - The adaptive testing strategy
  - Hints

How do I get setup?

- Requirements
- Using Gradle from command line
- Using your IDE of choice

Requirements

- MBT-module requirements
  - JDK 1.8 — https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
    - Set the JVM 1.8 as default one
  - R environment — https://www.r-project.org/
    - R packages
      - MCMCpack — https://cran.r-project.org/web/packages/MCMCpack/index.html
      - HDInterval — https://cran.r-project.org/web/packages/HDInterval/index.html

- MBT-module
  - Open source Git repository — https://github.com/SELab-unimi/mbt-module
  - Clone the repository
  - Execute `cd` into the repo folder (mbt-module)
  - Execute `git checkout no-rjava`

Command line — build + execution

- From the repo folder (mbt-module)
  - # bash

```
> ./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 *********
…
```

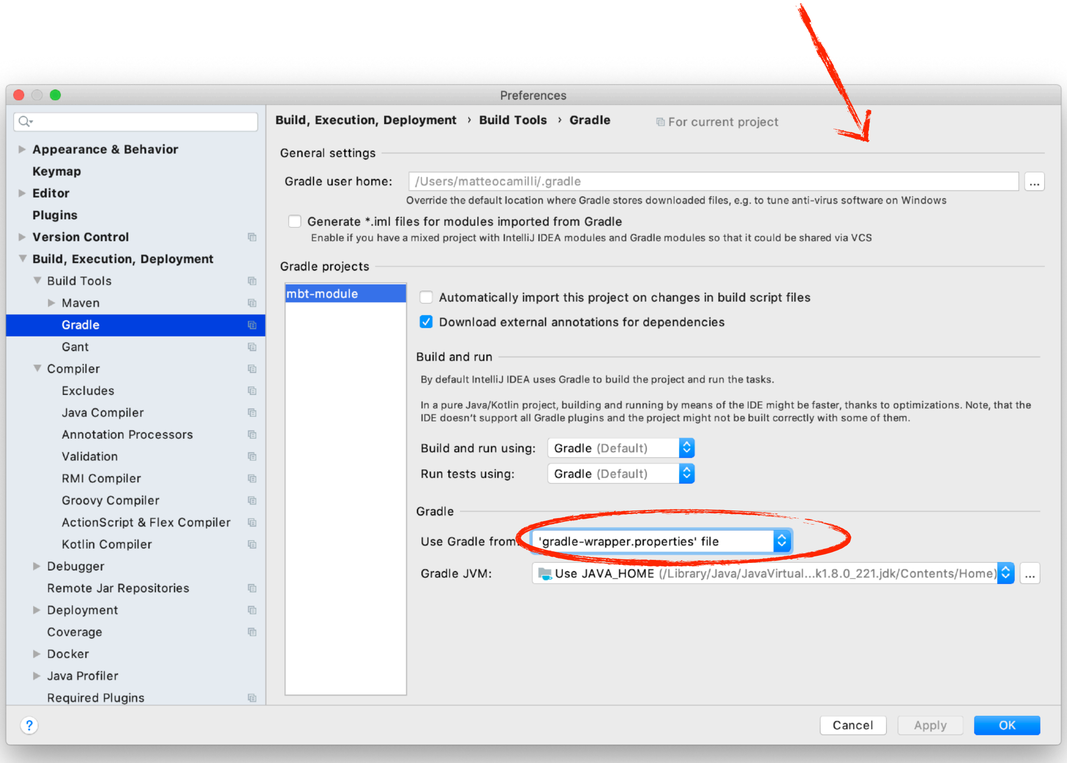
Import into the IDE

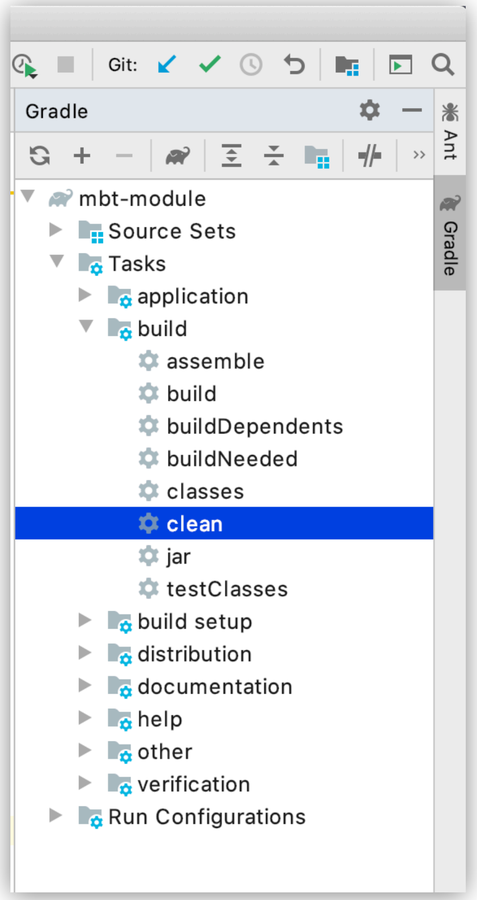
- Intellij Idea IDE
  - https://www.jetbrains.com/idea/
  - Ultimate edition needed (it includes the aspectJ plugin)
- Eclipse IDE
  - https://www.eclipse.org/downloads/
  - AspectJ Development Tools (AJDT) plugin (https://marketplace.eclipse.org/content/aspectj-development-tools)

- Once you have the IDE (with the aspectJ plugin)
  - Open/import an existing gradle project
  - Select the mbt-module directory

Build from the IDE

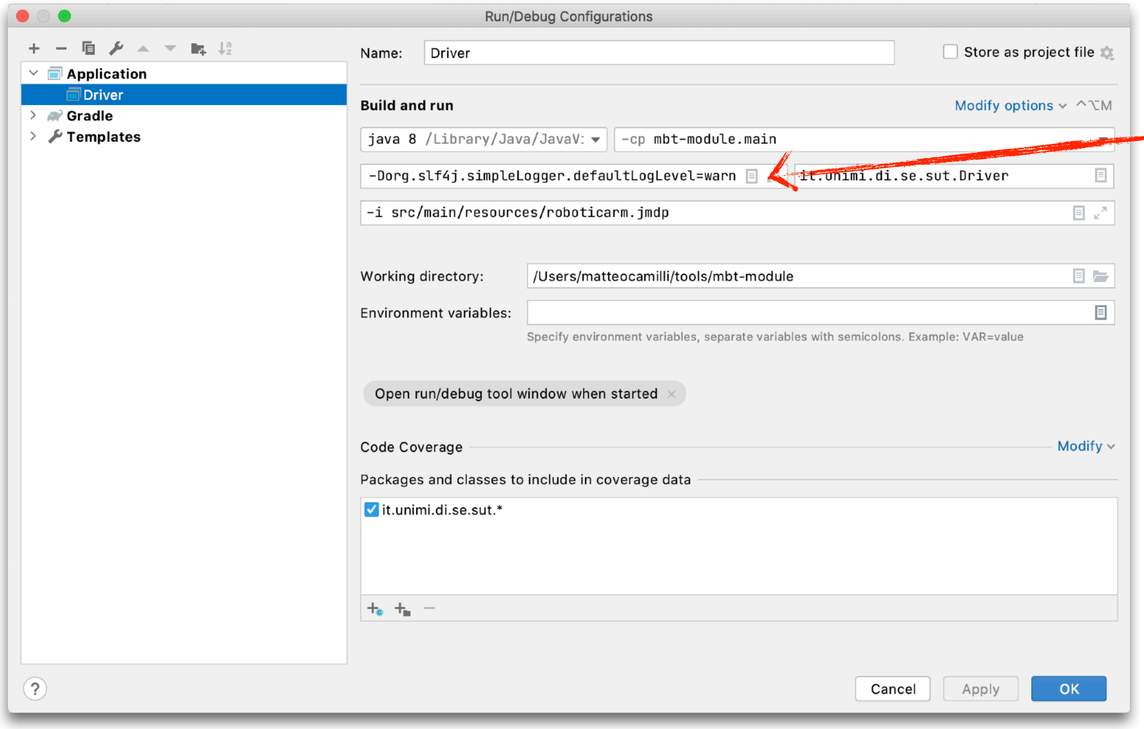
- Clean + build
  - Use the IDE to run **gradle** commands: **clean** and then **build**
  - Be sure that **gradle-wrapper** option is selected





Run from the IDE

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



VM options (taken from build.gradle)

-Dorg.slf4j.simpleLogger.defaultLogLevel=warn

Exercises

- Inspect and then run
- Try out different strategies
- Advanced: design/implement a new testing strategy

(1) 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]
  - What are 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 either from the command line or from the IDE
  - What is the outcome you get for each uncertain region?
  - Compute the HDR of each posterior by running the given command in a R terminal (MCMCpack + HDInterval libs needed)

(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 or equal to 0.01

- Compare the cost-effectiveness of the flat strategy vs the totally random one
  - Execute the **random** strategy (`Policy.RANDOM`) **10 times** and for each run:
    - Compute the HDR of **(S10, e1)**
    - Count **how many times** the computed bounds **do not meet** R1
  - Repeat this experiment with the **uncertainty flat** strategy (`Policy.UNCERTAINTY_FLAT`)
  - Which strategy is likely to be superior to detect violations?

(3) Implement an adaptive strategy

- Problem
  - All the testing strategies considered up to this point are built considering a **pre-computed** and **fixed** set of best policies maximizing the probability of reaching each region (in isolation)
  - However, the best policies are computed based on the prior knowledge and this might lead to suboptimal exploration in case the initial assumptions are wrong
- Idea — implement an adaptive strategy by following this guideline
  - Periodically (e.g., every "**sample size**" tests) update an "adaptive" **reward structure** that assigns rewards proportional to the **HDR magnitude**
  - The **reward** associated to arcs of each uncertain region should be **directly proportional** to the **HDR magnitude** of that region (i.e., the larger the region, the higher the reward)
  - Compute the **best policy** given the "adaptive" **reward structure**
  - Select the **actions** by following the **new best policy**
  - Repeat until **termination**
- Technical details
  - To implement and run this strategy you need to have the full framework available in the *master* branch of the repository
  - Checkout the *master* and follow the instructions in the README to setup your environment

(3) Implement an adaptive strategy — hints

- Hints
  - The testing strategies follow the strategy design pattern and extend the DecisionMaker class
  - The DecisionMaker receives information on the HDR magnitude of regions by means of the callback updateDistance invoked by the Monitor (see the DistanceDecisionMaker)

```
50      @Override
51      public void updateDistance(int stateIndex, double distance) {
52          hpdDistance.put(new IntegerState(stateIndex), distance);
53      }
```

DistanceDecisionMaker.java

  - The DecisionMaker interface should be extended with a new method updatePolicy called by the Monitor every sample-size observations

```
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                     }
```

Monitor.java

else updatePolicy call

License of these slides