Molr

A delegation framework for accelerator commissioning

Nachiappan V, TE-MPE-MS, CERN
# The problem

<table>
<thead>
<tr>
<th>Context</th>
<th>Problem statement</th>
<th>Requirements</th>
</tr>
</thead>
</table>
| AccTesting orchestrates the commissioning of CERN’s accelerators and its equipment systems, which involves running tests provided by various commissioning tools + analyzing their outcome | Develop a unified framework for integrating custom commissioning tools with AccTesting | • Increase automation in commissioning and analysis  
• Re-use code of existing (legacy) commissioning tools |
AccTesting

- Accepts user defined commissioning tests
- Schedules tests considering multiple constraints
- Provides UI to monitor test execution + analysis
Example of a typical commissioning test
Molr

A framework to:

- Easily integrate new commissioning tools
- Delegate execution of tests

Technical elements:

- Location transparency
- Dynamic dispatching of code
- Interactive control of executing code
Challenges

**API**
- Simple, but generic
  - Should be easy to integrate and deploy existing tests

**Control, I/O**
- Remote, but local-like
  - Provide input
  - Retrieve results
  - Control execution
  - Error propagation

**Scaling**
- Load distribution
  - Tens of thousands of tests are run during a typical commissioning period (typically after longer stops of an accelerator)
Solution?

Hire a summie!

CERN SUMMER STUDENT PROGRAMME: ACCELERATING SCIENTISTS

BRINGING STUDENTS TOGETHER SINCE 1962

Credits: Lillian Huang, CERN summie
Molr Architecture
Molr used by Developer

```java
@RunWithMole(IntegerFunctionMole.class)

public class Fibonacci implements Function<Integer, Integer>{
```
Molr used by Operator

CompletableFuture<RunMissionController<Integer>> futureController =
    mExecService.<Integer, Integer>runToCompletion("cern.molr.sample.Fibonacci",
        42, Integer.class, Integer.class);

try {
    RunMissionController<Integer> controller = futureController.get();
    CompletableFuture<Integer> futureResult = controller.getResult();
    return futureResult.get();
} catch (Exception e) {
    e.printStackTrace();  //logging is for children
    throw e;
}
Implementation

Tools

- Interfaces and Implementation: **Java**
- Communication & serialization: **Spring Boot**
- Infrastructure delegate: (Docker Swarm / Kubernetes / ?)

Highlights

- Functional programming patterns (borrowed from Haskell, exploiting Java 8)
- Strong static type safety (exploiting Java generics)
Use cases

Collimator commissioning

Applications for the Beam Interlock system
e.g.: closing the Beam permit frequency loop

Hardware Commissioning Steps

- Full system tests (with or without vacuum)
  - remove blocking of jaws
  - verification of collimator corner orientation (in the tunnel)
  - check low level control, jaw movement and position
  - sensors/switch response, including the verification of the position
    of the switch with respect of the end stop (only relative position
    needed here)
  - check temperature sensors
  - LVDT calibration and resolver check
  - check interlock chain
  - check communication (from top level [CCC-room application to
    collimator])
  - check water tightness/adjust flow-rate

Fig. 3: Architecture of the Beam Interlock System
Conclusion

- First prototyping confirms Molr to be a viable option for integrating standalone commissioning tests into the operational AccTesting framework
  - Allows for required degree of flexibility and interaction with the executed tests
  - Allows to increase the coherency of commissioning campaigns and therefore the overall dependability of the protection systems at the time of machine re-starts
- Next step: Validation and integration of first commissioning steps via Molr for upcoming commissioning campaign at the end of the next Year End Technical Stop (YETS)
- Molr is open source! https://github.com/molr
Thank you!

Thanks to Markus, Marc-Antoine & the rest of my team for helping with this presentation :}