

Jenkins for FloBI – A Use Case: Jenkins & Robotics



Florian Lier, Johannes Wienke,
Sebastian Wrede

CITEC / CoR-Lab, Bielefeld University,
Germany

<http://www.cit-ec.de> – <http://cor-lab.org>

@jenkinsconf



Who are we?



Johannes Wienke

PhD Student
Cognitive Systems Engineering Group
CoR-Lab/CITEC
Bielefeld University, Germany

Research Focus

- Software engineering in robotics
- Middleware development
- Jenkins support



Outline



- **The Motivation**
Integrated, Replicable & Tested Research Systems
- **The Journey**
Integration of Research Software Distributions
 - Supporting C++ Component Development
 - Software Distribution Management
- **Finally**
Functional Testing in Robotics
 - FSMT: State Machine for Integration Testing
 - Simulations vs. Reality
- **The Rest**
Wrap-Up



The Motivation

INTEGRATED, REPLICABLE & TESTED RESEARCH SYSTEMS

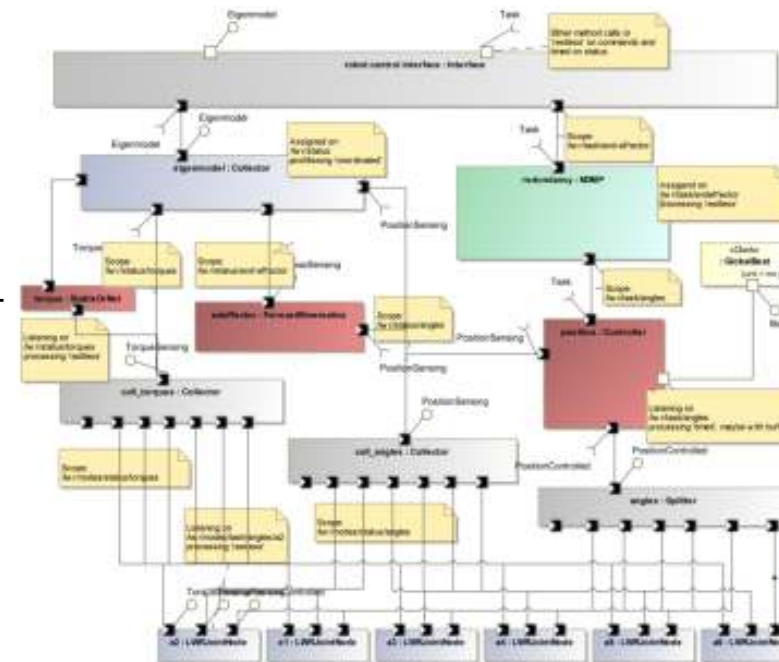
Research Context – Systems



Development Situation: The Systems



- Complexity factors
 - Multitude of different component types included
 - Sensory processing
 - Data management
 - Machine learning
 - Infrastructure tools
 - Multitude of programming languages, especially C++
 - Evolved and adapted to personal needs
 - Reuse of external software
 - Not all required disciplines in house
- Automated testing is hard to achieve
 - Hardware in the loop
 - Interaction with humans and environment
 - Nondeterministic behavior

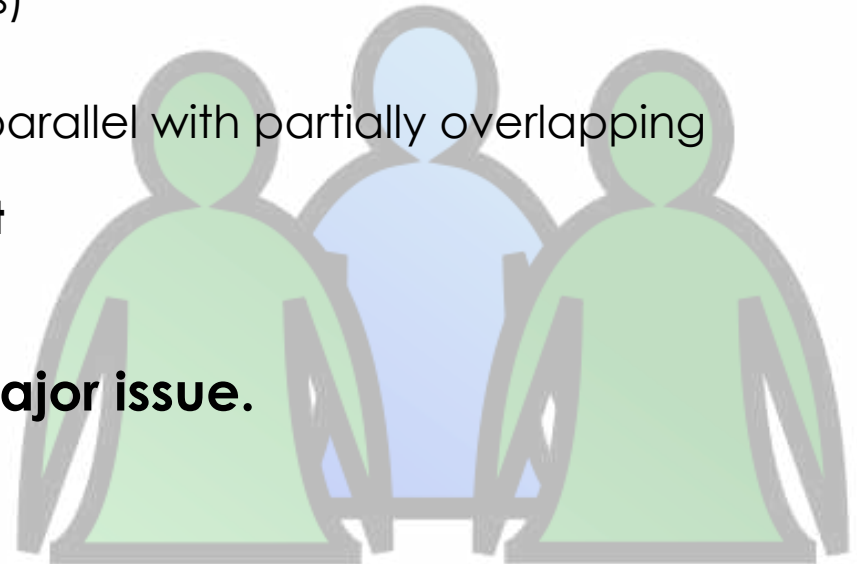


Development Situation: The People



- Highly **interdisciplinary**
 - Many people without formal software development education
 - System integration is a new challenge for many
- Most developers are PhD students
 - Main interest: the own PhD thesis
 - **Designing and maintaining reusable software needs conviction**
 - Limited time at university (3 – 5 years)
- Research project structure
 - Multiple research project active in parallel with partially overlapping goals or methods
 - **Decentralized project management**
 - Distributed teams

Continuity and sustainability are a major issue.



Integrated, Replicable & Tested Research Systems



We need integrated, replicable & tested systems to succeed with the research work.

- **Software issues** are a **major slowdown** in research
 - Needed: **Simplified integration**
- But also **reliability of the integrated system** w.r.t. the task
 - Needed: **Functional testing**
- → Software engineering plays an important role in robotics research

How to achieve this?



The Journey

INTEGRATION OF RESEARCH SOFTWARE DISTRIBUTIONS

The Toolkit Idea



- Provide a well-maintained core of software packages available in a **ready-to-use distribution**
 - Immediately available with minimal setup effort
 - Single software basis for all experiments and demonstrators
- However
 - Requires that **people maintain** their software appropriately
 - We need to **convince** them by providing benefits

We try to implement a process that **provides a high level of convenient automation and development support based on Jenkins.**

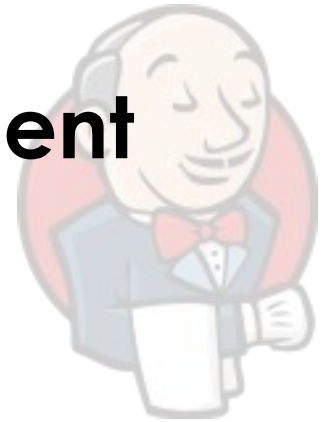


The Journey

INTEGRATION OF RESEARCH SOFTWARE DISTRIBUTIONS

Supporting C++ Component Development

Supporting Component Development



Outline of our talk at JUC 2011

- C++ project setup:
 - **Relocatable projects** (with CMake)
 - Building on diverse nodes
 - Code analysis and metrics
- Shared resources on build slaves
- Common CMake job template
- Lessons learned with C++ and Jenkins
- Off-site development

Most important

Allows use of artifact copying between jobs
→ Use Jenkins scheduler and parallel builds
→ Prevent global shared state (install prefix)

Building individual C++ (native) projects with Jenkins was a major issue. Solution is quite stable now.

General Job Structure



- Main build and test on different Linux flavors
- Separate jobs for windows and Mac OS
 - Too many differences
- Separate packaging (e.g. DEB)
 - Different compiler settings
- Static code analysis in extra jobs
 - Reduce usual build time
 - Includes API doc generation
- Merge simulators for testing development branches in trunk version
- Further task-specific jobs

Linux Main Build Job Setup

- **Multiconf projects:**
 - Linux slaves as axis
- **SCM:**
 - Git or SVN:
 - Wipe out workspace
- **Triggers:**
 - Upstream-Downstream
 - SCM
- **Build-timeout plugin**
 - If required
- **Copy Artifacts plugin**
 - Fetch upstream dependencies
 - Requires relocatable projects
- **Bash build steps**
 - Extract and prepare upstream dependencies
 - Build and test project
 - Prepare reports
- **Warnings plugin**
 - Report compiler warning
- **Archive & fingerprint**
- **Report publishing**
 - JUnit format for tests
 - Cobertura plugin for coverage
- **Email-ext plugin**
 - Still failing + more details
- **Mail Upstream Committers plugin**
 - Check dependencies



Further Job Configuration Details



Package Jobs

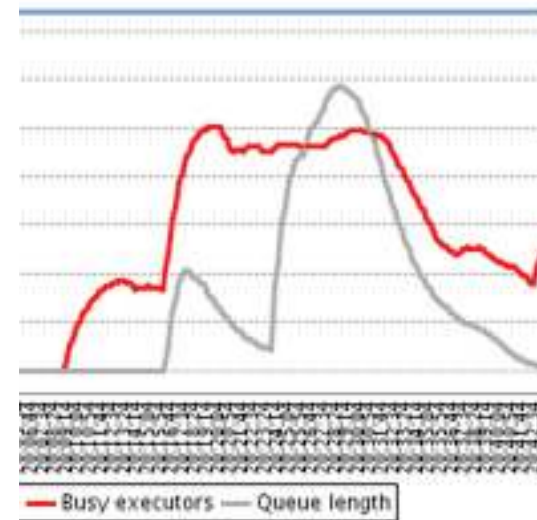
- For Debian packages
 - Lintian warnings parser for Warnings plugin
- Publish over SSH plugin
 - Upload to our package server

Static Analysis Jobs

- Task Scanner plugin
 - Measure number of TODO tags
- Warnings plugin
 - Find doxygen errors
- HTML Publisher plugin
 - Publish API doc
- SLOCCount plugin
- Cppcheck plugin
- Publish over SSH plugin
 - Send API doc to server

Job Setup Summary

- C++ setup for usual development jobs pretty stable
- Sufficient feature support integrated
- Scalability is limited:
 - Parallel build on multiple different linux distros
 - Check against different upstream versions
 - Check that users can install software without problems
 - Job/node scheduling is limited:
 - Artifacts mostly bound to a specific node
 - → Check against different distros less frequent
- Configuration aspects shared by many jobs: increased maintenance effort (→ next section)
- User feedback is mostly good
 - Initial reservations for setup overhead
 - Value recognized afterwards
 - Intrinsic complexity of C++ projects causes some ongoing maintenance efforts: many subtle things can cause errors



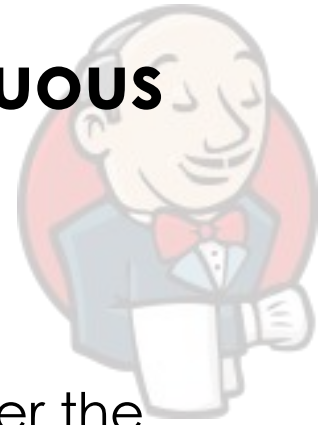


The Journey

INTEGRATION OF RESEARCH SOFTWARE DISTRIBUTIONS

Software Distribution Management

A Generator-based Approach Continuous Distribution Integration & Delivery



Idea

Let Jenkins build and continuously test, deploy and deliver the whole toolkit distribution.

However

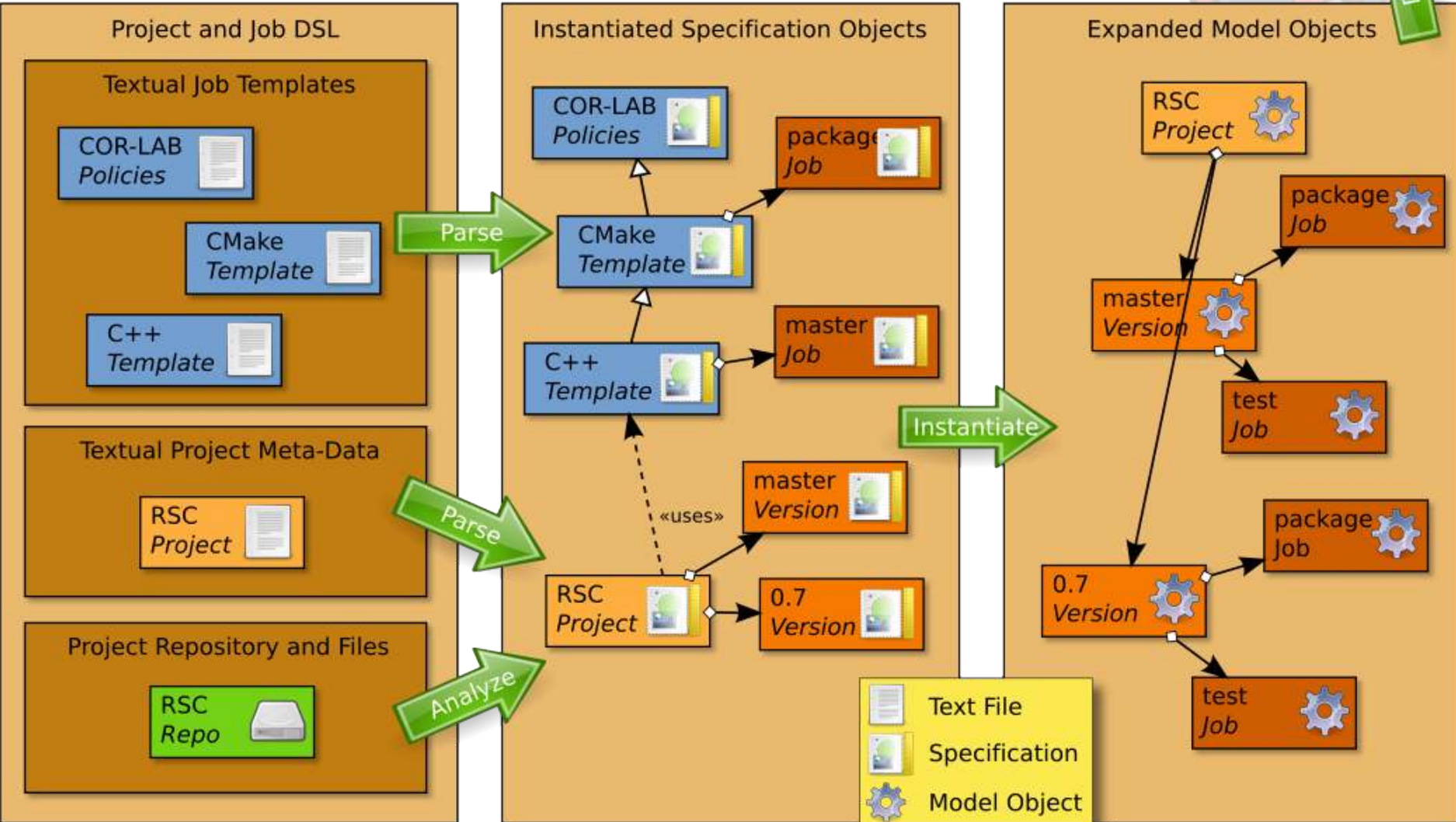
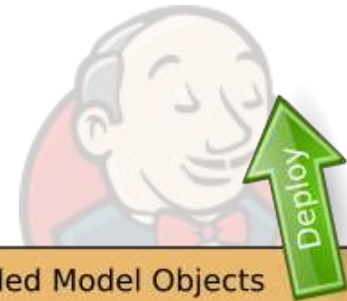
Manual management of all different job types for each component does not scale → maintenance nightmare.

Solution

- Specify a minimum of software information in a DSL
- Automatically generate Jenkins jobs

We have prototypical tool support for this.

Job Tool Realization



Job Tool Realization



Project and Job DSL

Textual Job Templates

COR-LAB
Policies



CMake
Template

C++
Template



Textual Project Meta

RSC
Project



Project Repository and Files

RSC
Repo



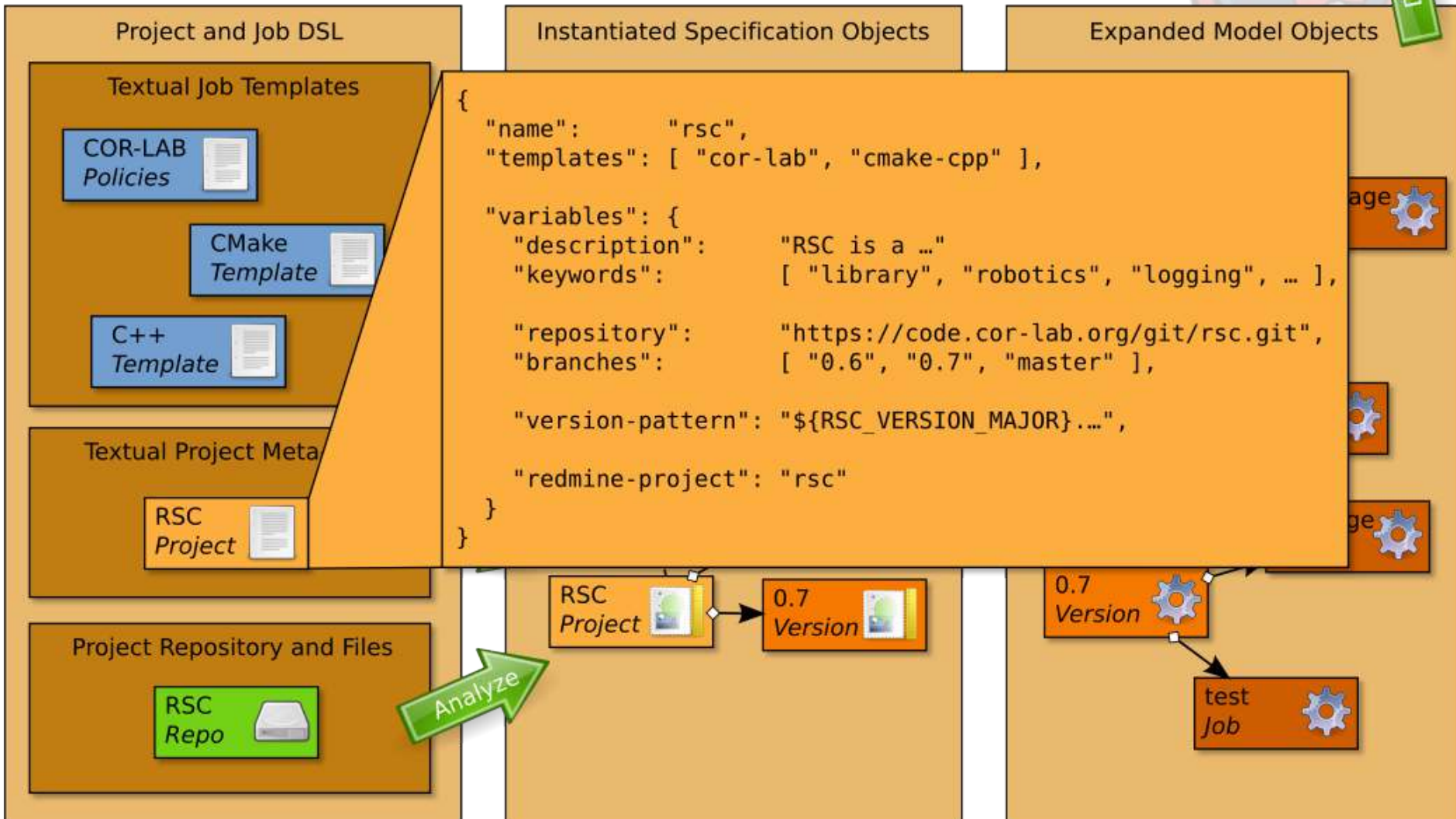
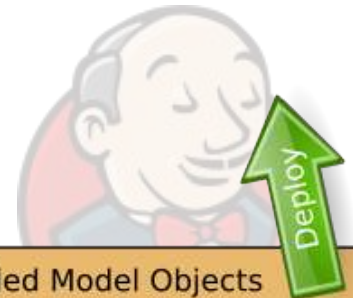
```
{
  "name": "cmake-cpp",
  "inherit": [ "cor-lab" ],

  "variables": {
    "kind": "matrix",
    "tasks.pattern": [ "**/*.h*", "**/*.c*", ... ],
    "main.cmake.targets": [ "test", "coverage", "package" ],
    ...
  },

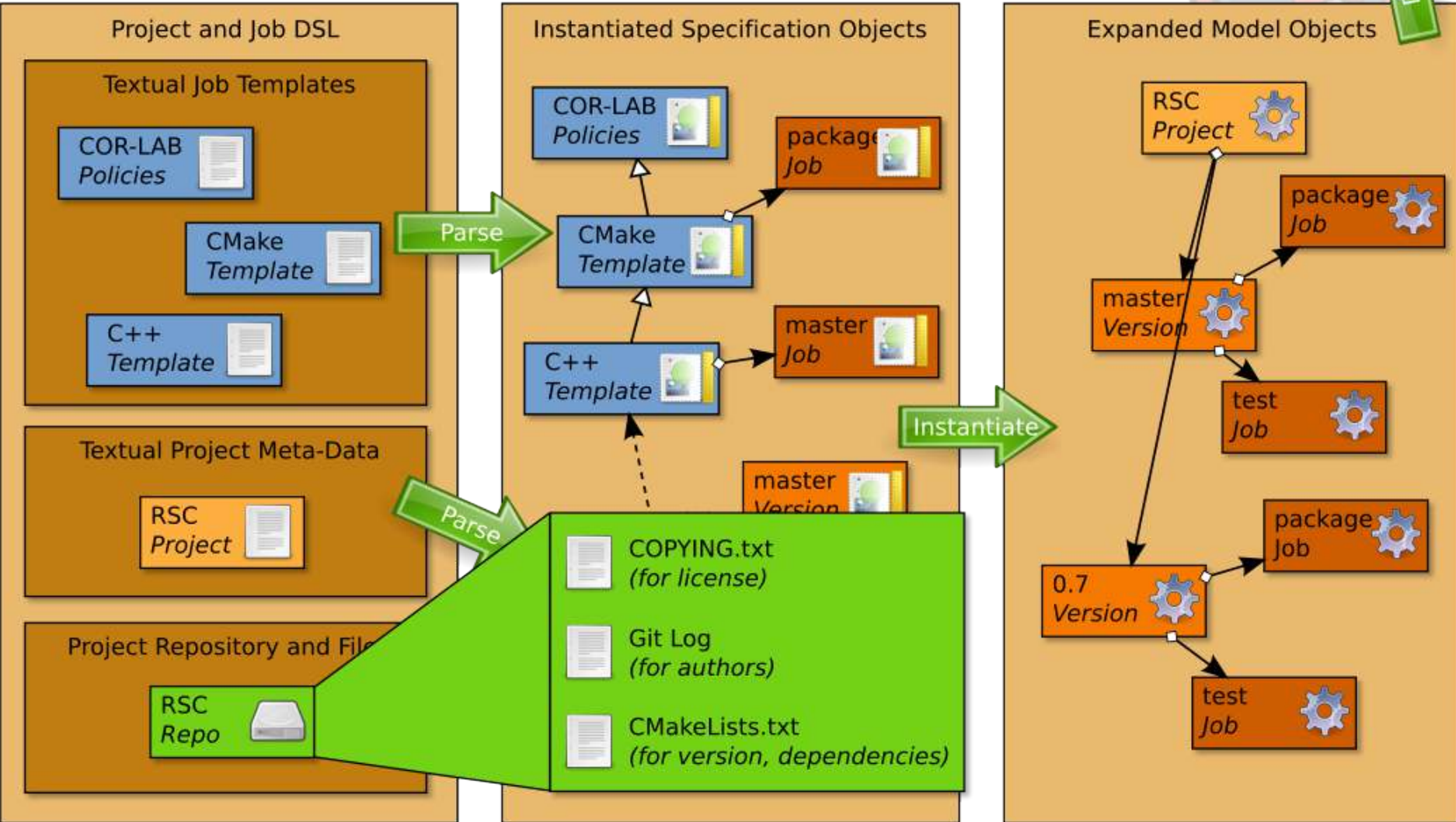
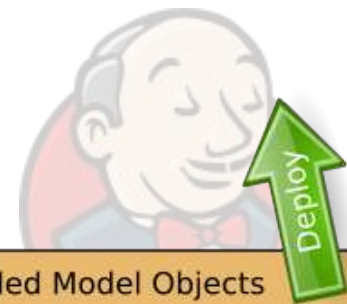
  "aspects": [
    {
      "name": "cmake-cpp.cmake/unix",
      "filter": "unix",
      "aspect": "cmake/unix",
      "variables": {
        "aspect.cmake.targets": "${cmake.targets}"
      }
    }, ... ]

  "jobs": [
    {
      "name": "",
      "tags": [ "architecture-dependent", "gcc", "unix", "linux" ],
      "variables": {
        "cmake.options": [ "CPACK_GENERATOR='TGZ;ZIP'",
                          "CMAKE_BUILD_TYPE=coverage" ],
        "cmake.targets": [ "@{main.cmake.targets}" ]
      }
    }, ... ]
  }
```

Job Tool Realization



Job Tool Realization






Parse

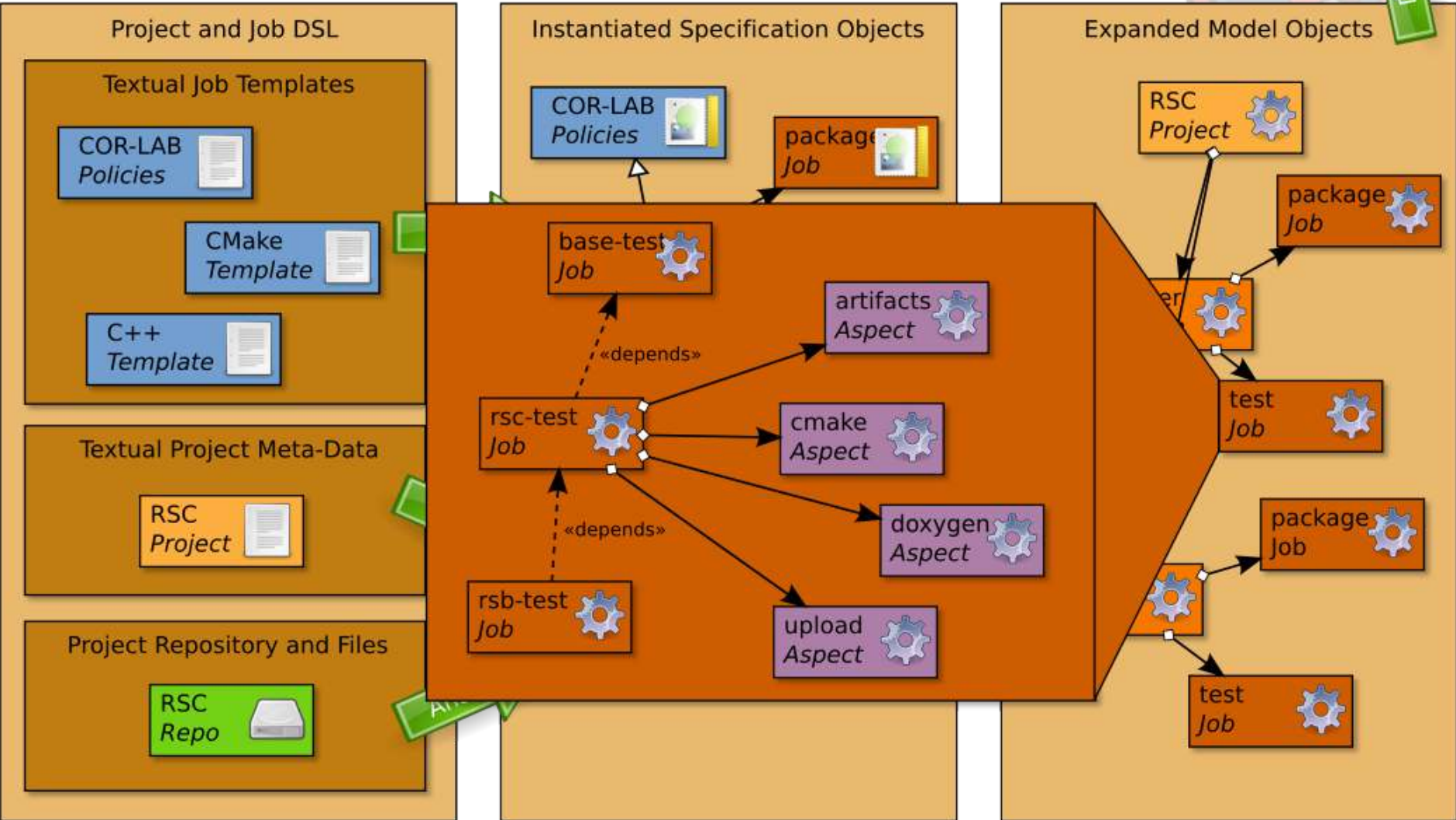
Parse

Instantiate

Deploy

-  COPYING.txt (for license)
-  Git Log (for authors)
-  CMakeLists.txt (for version, dependencies)

Job Tool Realization



Job Orchestration for Distributions



[cca-trunk-toolkit-nightly](#)

[continuumkinematics-trunk-toolkit-nightly](#)

[fsmt-trunk-toolkit-nightly](#)

[icl-trunk-toolkit-nightly](#)

[kdl-master-toolkit-nightly](#)

[libamarsi-master-toolkit-nightly](#)

[libximu-trunk-toolkit-nightly](#)

[nemobench-trunk-toolkit-nightly](#)

[nemomath-trunk-toolkit-nightly](#)

[openrave-master-toolkit-nightly](#)

[pyscxml-master-toolkit-nightly](#)

[rci-lwr-rave-trunk-toolkit-nightly](#)

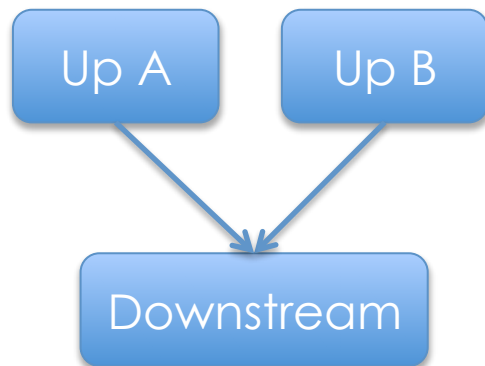
Problem

Upstream-downstream triggers do not work well in this situation

- No way to ensure dependency on multiple upstream projects
 - Temporary build errors
 - Jobs triggered too often

Solution

- Build flow plugin
 - Job tool derives a graph for the build order with parallel fragments
 - Build flow plugin executes this graph



Build Flow Graph for Toolkit Nightly

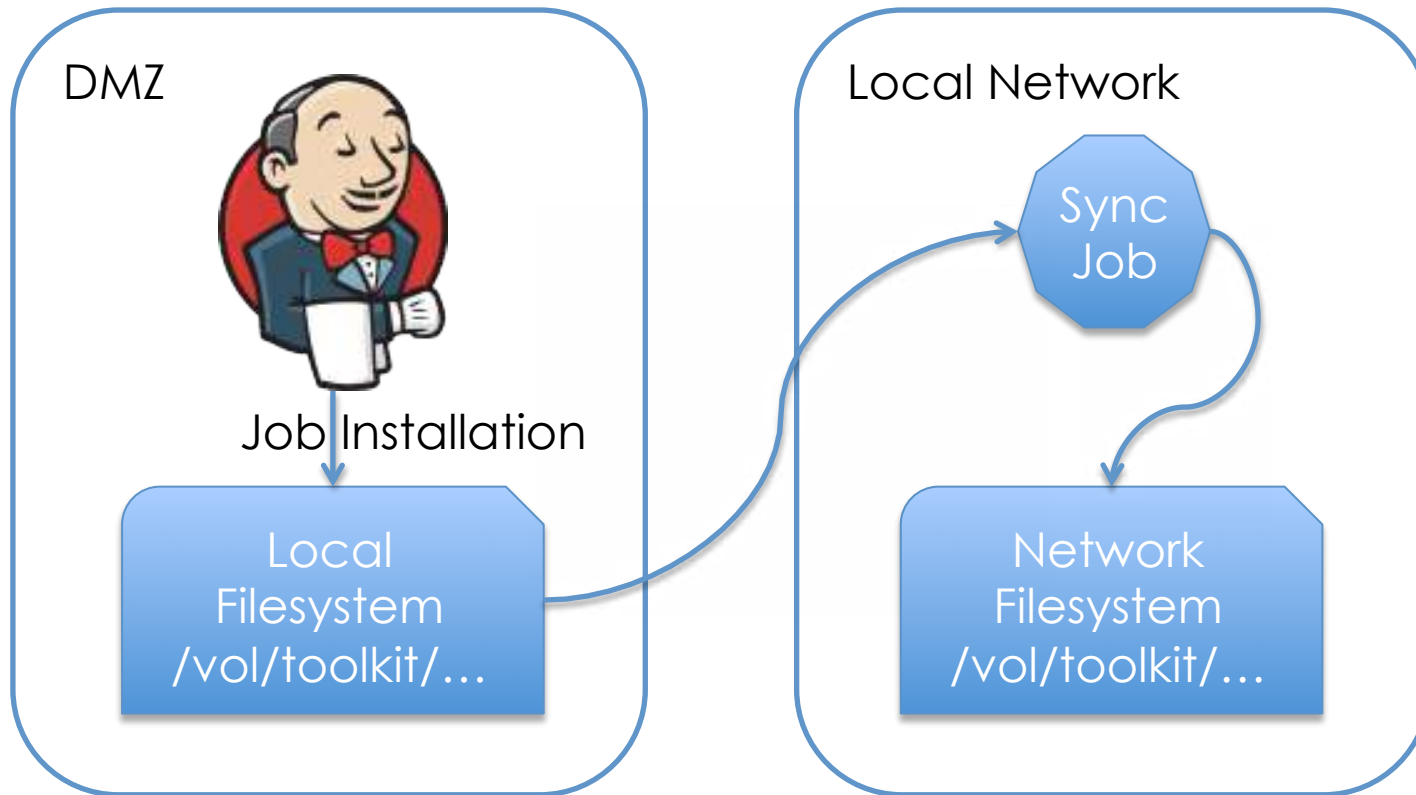


Some Issues

- Rendering slow and not meant for big graphs, only works in Firefox
- Assessing status of all jobs at end of flow is cumbersome



Continuous Delivery / Deployment



Toolkit distribution available immediately in our network for end users

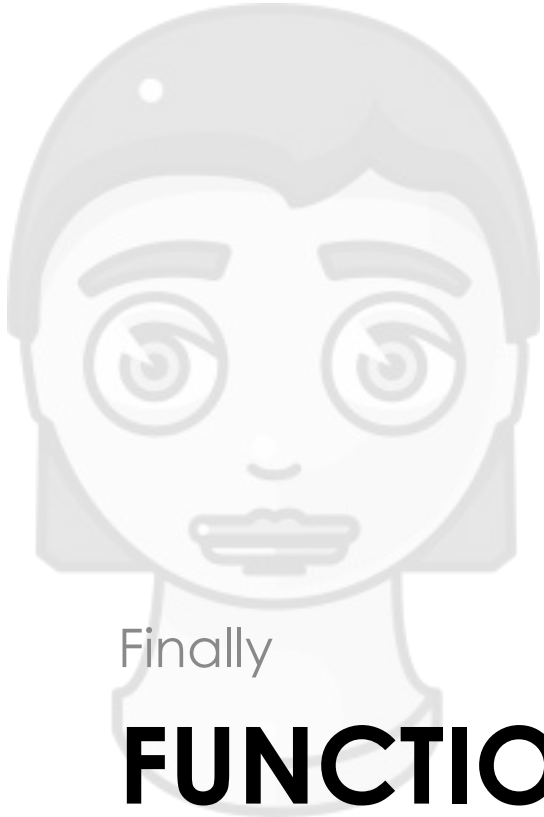
Job Tool Summary

Using a DSL-based project descriptions and a job generation approach results in

- Reduced maintenance
 - Similar changes across jobs can be automated
 - No duplicated information to keep consistent
 - Only a minimum of manual information is required
- More consistent job setups
- Easy replication of the setup to other Jenkins instances
- Survives Jenkins ;)

Achieves required scalability for large setups





Finally

FUNCTIONAL TESTING IN ROBOTICS

Systematic Functional Testing



Functional testing is a crucial part for robotics in order to verify correct behavior of robot systems

Functional System Testing Issues

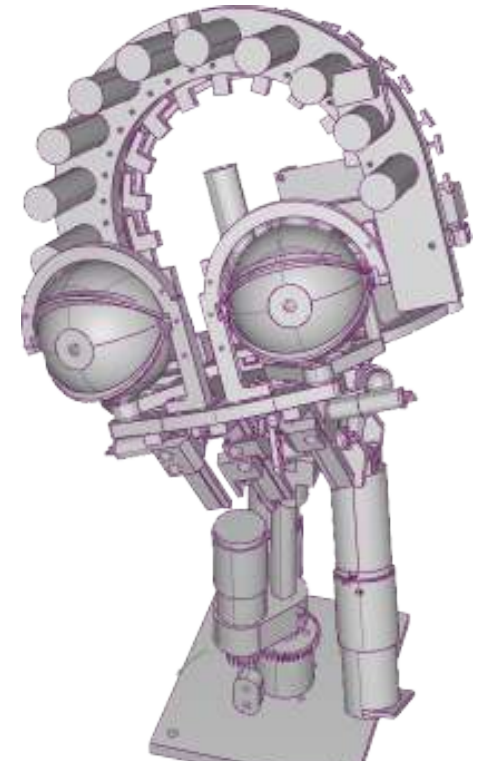
- Often **fuzzy metrics** of test success
 - What is an acceptable time to solve a task? People have different opinions.
- System tests are mostly carried out **manually**
 - Infrequent, high effort, hard to replicate, not comparable
- Component **execution order** and **timing** are important
 - Dependencies / order needs to be respected
 - Inconsistent timing may result in different results
- **Hardware in the loop**

Dealing with Hardware in the Loop

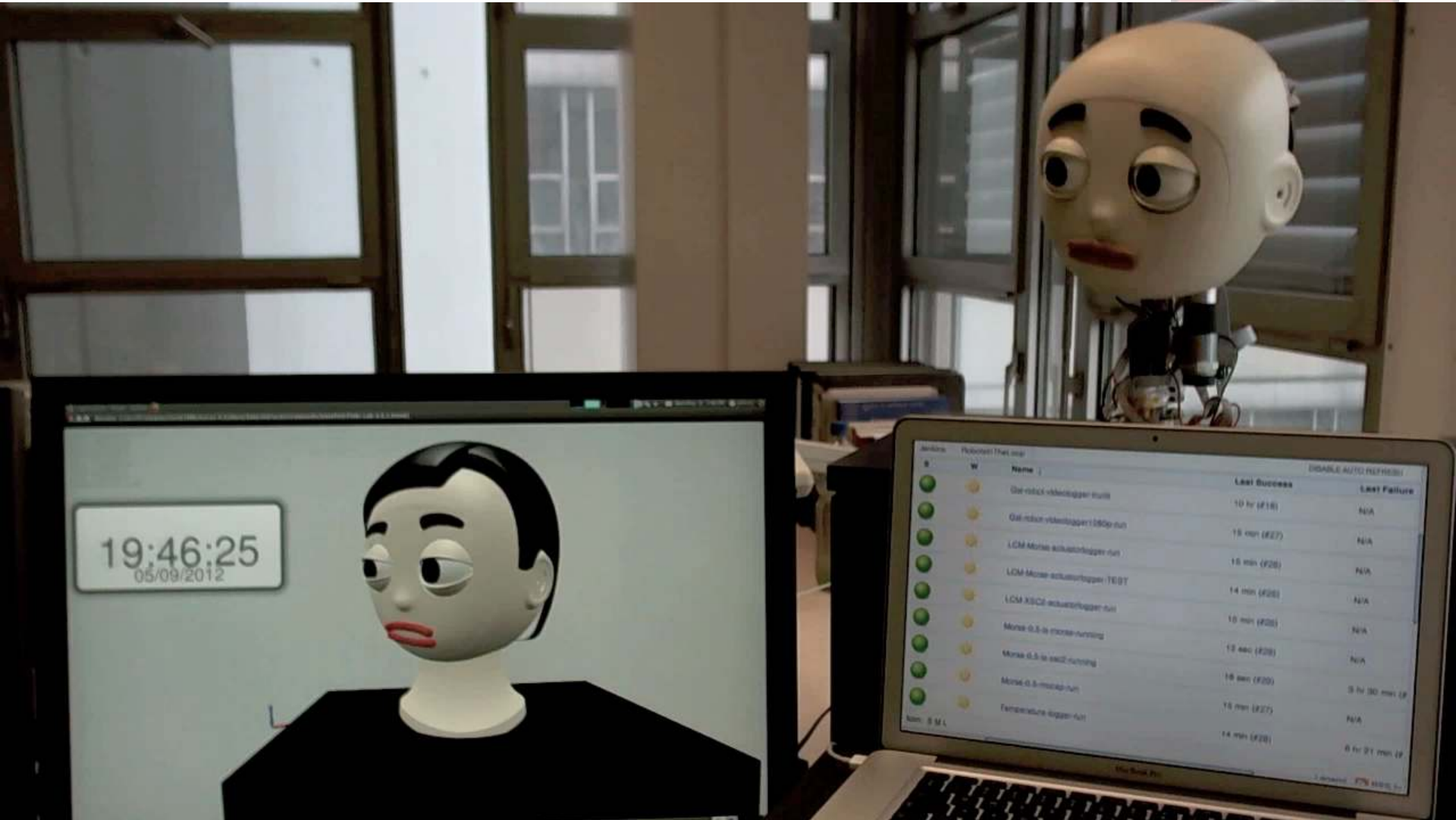
- Hardware is inherently a part of robot systems
- Issues for development:
 - Hardware is costly and mutually exclusive usage is required
 - Can easily be damaged
 - Sometimes needs maintenance
 - Hard to integrate and automate in general

Simulation is a commonly applied technique.

How accurate can simulation be?



Simulation vs. Reality

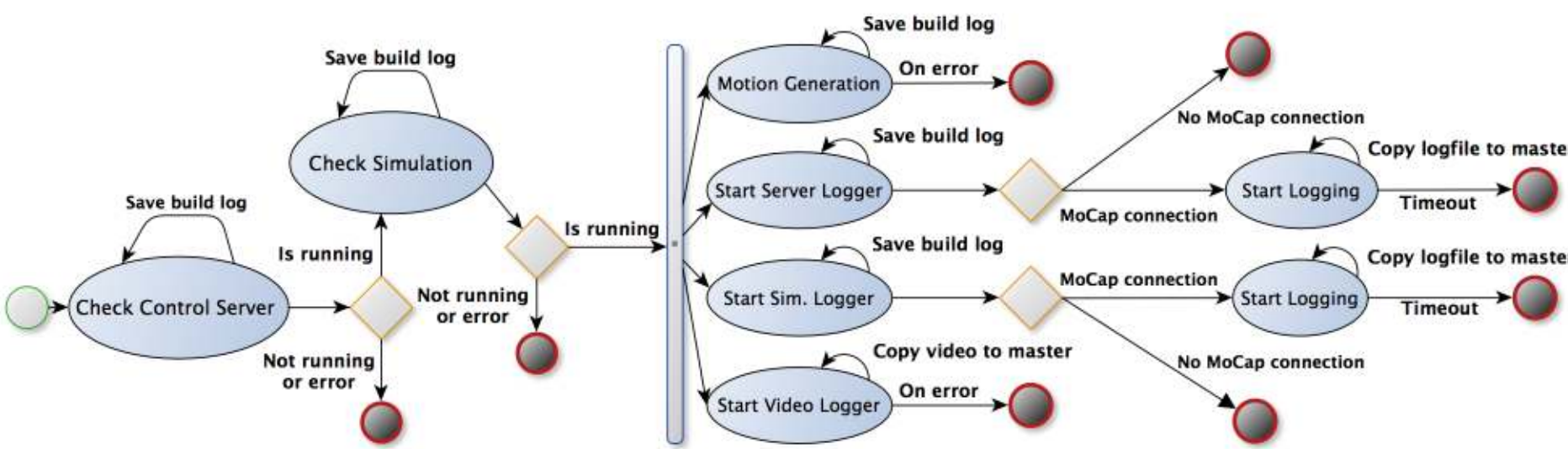


Test Component Orchestration in Jenkins

As Jenkins is used for automation:
model directly in Jenkins

- Starting components
- Trigger test execution
- Start logging components

		Gst-robot-videologger-trunk
		Gst-robot-videologger1080p-run
		LCM-Morse-actuatorlogger-run
		LCM-XSC2-actuatorlogger-run
		Morse-0.5-is-morse-running
		Morse-0.5-is-xsc2-running
		Morse-0.5-mocap-run
		Temperature-logger-run



Test Component Orchestration in Jenkins



Issues

Jenkins was probably never meant for this:

- Job coordination is hard to manage
 - Build flow could improve this
- Timing cannot be exactly defined
- Multiple jobs:
 - Scalability & Maintenance
 - Overview hard assess
- Clean up actions need to be done manually

However

Jenkins still provides benefits

- SCM checkout & triggering
- Notifications
- Artifact archiving
- Build history
- Visibility as a server
- Multiconf projects for different test environments

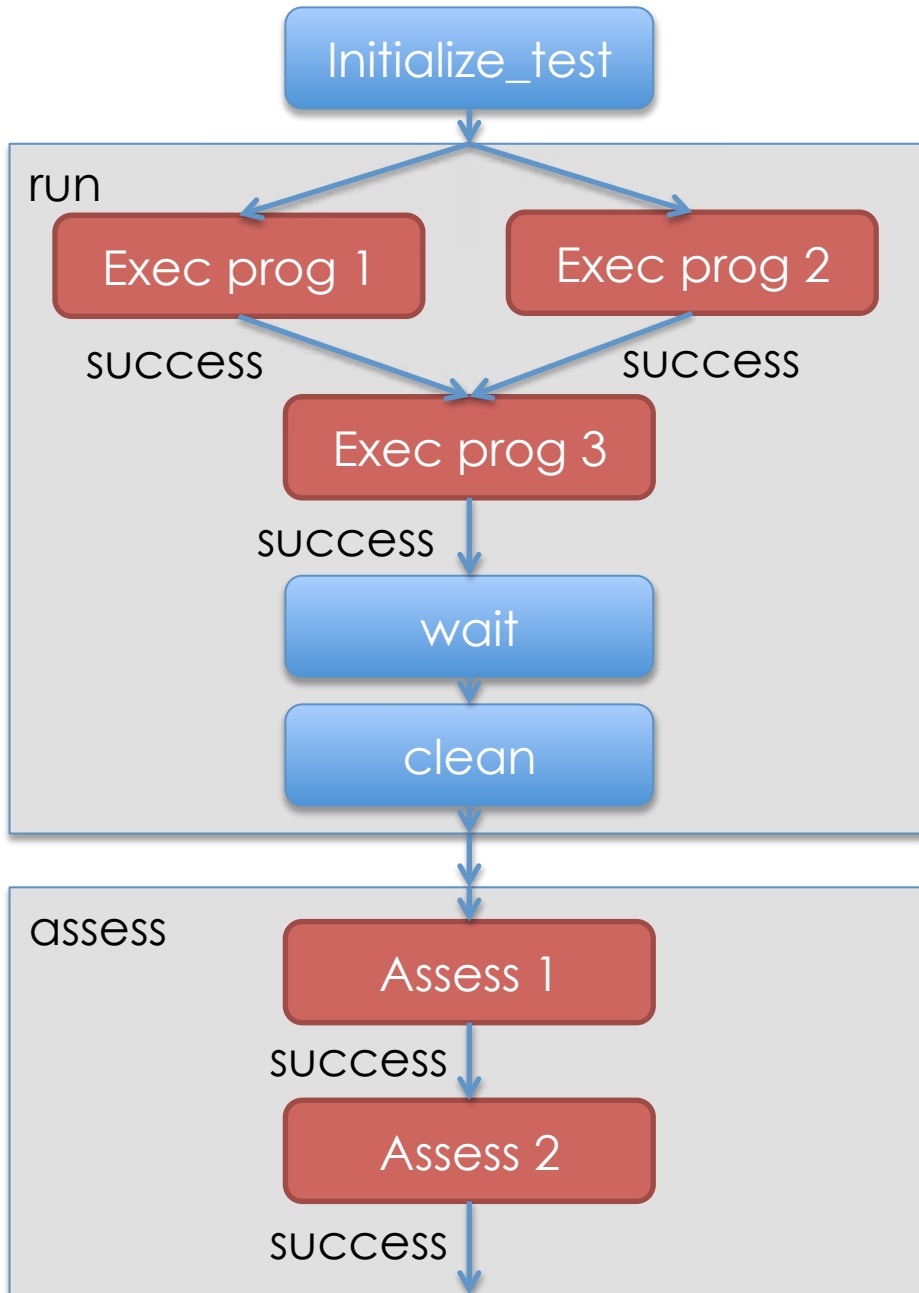
Simulation works; Jenkins setup too complicated for functional tests

FSMT: Finite-State-Machine-based Testing



Solution outside Jenkins required but still **usable in Jenkins**.

- Enable developers to design structured and replicable tests
 - Transparent and deterministic orchestration, repeatable and comparable
 - Explicit environment setup and configuration
 - Automatic system start-up and tear-down
 - Explicit verification of start-up
 - Easy to set up
- General purpose solution
 - Different robot systems
 - Multi-platform and -language
- Integration into CI server to fully automate testing



Method



Finite State Machines allow to define deterministic execution orders

Blue states: FSMT-defined standard execution phases

Red states: User-defined component start either for the system launch or for assessment

Transitions: User-defined criteria to check success of previous state

SCXML Test Specification

```
<scxml xmlns="http://www.w3.org/2005/07/scxml" xmlns:my_ns="http://my_namespace.com" id="morse_startup" initial="initialise_test" version="1.0">
  <datamodel>
    <data id="environment">
      <variable val="31232" var="ROS_PORT" />
      <variable val="http://localhost:31232" var="ROS_MASTER_URI" />
    </data>
    <data id="hosts">
      <hostinfo ip="127.0.0.1" name="localhost" />
    </data>
    <data id="component_bundle">
      <component val="roscore">
        <command val="roscore -p $ROS_PORT" />
        <path val="/opt/ros/groovy/bin/" />
        <executionHost val="localhost" />
        <checkExecution val="True">
          <checkType blocking="True" criteria="" ongoing="True" timeout="10" val="pid" />
          <checkType blocking="True" criteria="started core service" ongoing="False" timeout="30" val="stdout" />
        </checkExecution>
      </component>
      <component val="collect-logs">
        <command val="ls" />
        <path val="/bin/" />
        <executionHost val="localhost" />
      </component>
    </data>
  </datamodel>
  ...
</scxml>
```

Too complicated to write down



INI Test Specification

```
[environment]
ROS_PORT=31232
ROS_MASTER_URI=http://localhost:31232
```

```
[component-roscore]
name = roscore
command = roscore -p $ROS_PORT
path = /opt/ros/groovy/bin/
execution_host = localhost
check_execution = True
check_type = pid,stdout
timeout = 10,30
blocking = True,True
ongoing = True,False
criteria = ,started core service
```

...

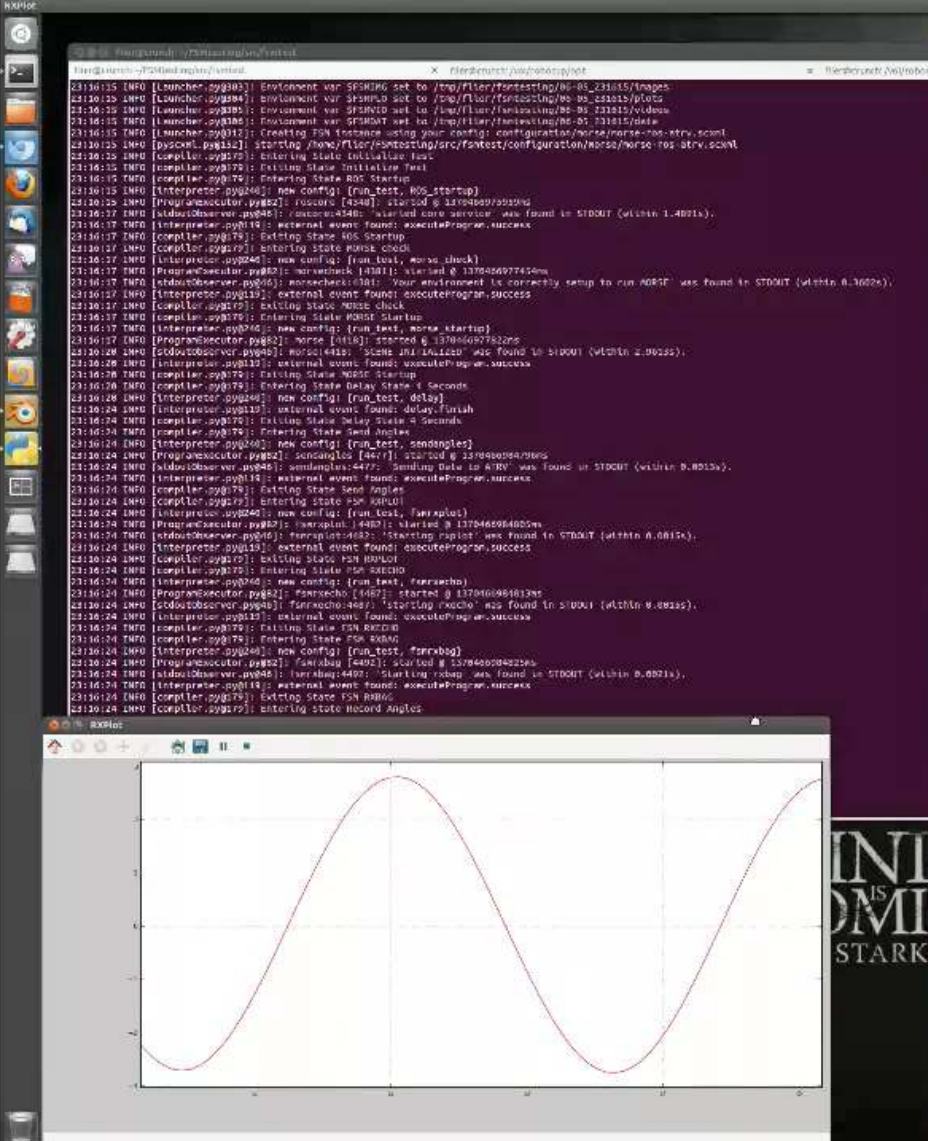
```
[component-collector]
name = collect-logs
command = ls
path = /opt/fsmt-tests/analyze-logs.py
execution_host = localhost
check_execution = True
check_type = pid
timeout = 2
blocking = True
ongoing = False
criteria =

[run]
name = morse_startup
namespace = http://my_namespace.com
run_order = ('roscore','morse','ffmpeg','nav-
stack','patroller','topic-logger'),
run_execution_duration = 20
result_assessment_order = ('collect-logs',),
result_assessment_execution_duration = 1
```





Proof of Concept Video



```
23:16:15 INFO [Launcher.py8803]: Environment var SPHINX set to /tmp/rlor/forecasting/06-05_231615/images
23:16:15 INFO [Launcher.py8804]: Environment var SPHINX set to /tmp/rlor/forecasting/06-05_231615/pilot
23:16:15 INFO [Launcher.py8805]: Environment var SPHINX set to /tmp/rlor/forecasting/06-05_231615/rover
23:16:15 INFO [Launcher.py8813]: Creating PSN instance using your config: configuration/rover/rover-no-ctrl.scml
23:16:15 INFO [pyCont.py8821]: Starting /home/rlor/forecasting/arc/psn/test/configuration/rover/rover-no-ctrl.scml
23:16:15 INFO [compilor.py8870]: Entering State Initialize Test
23:16:15 INFO [compilor.py8871]: Entering State ROS Startup
23:16:15 INFO [Interpreter.py8865]: new config: [run_text, ROS_Startup]
23:16:15 INFO [ProgramExecutor.py882]: roscore [4248]: started @ 13704609795594
23:16:17 INFO [stdexecServer.py846]: roscore:4248: started core service: was found in STDMIT (within 1.4001s).
23:16:17 INFO [Interpreter.py8876]: external event found: executeProgram.success
23:16:17 INFO [compilor.py8873]: Entering State ROS Startup
23:16:17 INFO [compilor.py8873]: Entering State ROS Check
23:16:17 INFO [Interpreter.py8846]: new config: [run_text, ROS_Check]
23:16:17 INFO [ProgramExecutor.py882]: roscheck [4248]: started @ 13704609774594
23:16:17 INFO [stdexecServer.py846]: roscheck:4248: started core service: was found in STDMIT (within 0.3602s).
23:16:17 INFO [Interpreter.py8849]: external event found: executeProgram.success
23:16:17 INFO [compilor.py8873]: Entering State ROS Startup
23:16:17 INFO [Interpreter.py8846]: new config: [run_text, ROS_Startup]
23:16:17 INFO [ProgramExecutor.py882]: ROS [4248]: started @ 13704609783295
23:16:18 INFO [stdexecServer.py846]: ROS [4248]: Started JPL-Python: was found in STDMIT (within 2.0022s).
23:16:18 INFO [Interpreter.py8849]: external event found: executeProgram.success
23:16:18 INFO [compilor.py8873]: Entering State ROS Startup
23:16:18 INFO [compilor.py8873]: Entering State Delay State 4 Seconds
23:16:18 INFO [Interpreter.py8849]: new config: [run_text, delay]
23:16:18 INFO [ProgramExecutor.py882]: delay [4248]: started @ 13704609804594
23:16:18 INFO [stdexecServer.py846]: delay:4248: started core service: was found in STDMIT (within 0.8815s).
23:16:18 INFO [Interpreter.py8849]: external event found: executeProgram.success
23:16:18 INFO [compilor.py8873]: Entering State Delay State 4 Seconds
23:16:18 INFO [compilor.py8873]: Entering State Send Angles
23:16:18 INFO [Interpreter.py8849]: new config: [run_text, sendAngles]
23:16:18 INFO [ProgramExecutor.py882]: sendAngles [4248]: started @ 13704609819594
23:16:18 INFO [stdexecServer.py846]: sendAngles:4248: Started Delay to RV: was found in STDMIT (within 0.8015s).
23:16:18 INFO [compilor.py8873]: Entering State Send Angles
23:16:18 INFO [compilor.py8873]: Entering State PSN KRL01
23:16:18 INFO [Interpreter.py8849]: new config: [run_text, KRL01]
23:16:18 INFO [ProgramExecutor.py882]: KRL01 [4248]: started @ 13704609840594
23:16:18 INFO [stdexecServer.py846]: KRL01:4248: Started KRL01: was found in STDMIT (within 0.8815s).
23:16:18 INFO [Interpreter.py8849]: external event found: executeProgram.success
23:16:18 INFO [compilor.py8873]: Entering State PSN KRL01
23:16:18 INFO [Interpreter.py8846]: new config: [run_text, KRL01]
23:16:18 INFO [ProgramExecutor.py882]: KRL01 [4248]: started @ 13704609861594
23:16:18 INFO [stdexecServer.py846]: KRL01:4248: Started KRL01: was found in STDMIT (within 0.8822s).
23:16:18 INFO [Interpreter.py8849]: external event found: executeProgram.success
23:16:18 INFO [compilor.py8873]: Entering State PSN KRL01
23:16:18 INFO [compilor.py8873]: Entering State PSN KRL01
23:16:18 INFO [Interpreter.py8846]: new config: [run_text, KRL01]
23:16:18 INFO [ProgramExecutor.py882]: KRL01 [4248]: started @ 13704609882594
23:16:18 INFO [stdexecServer.py846]: KRL01:4248: Started KRL01: was found in STDMIT (within 0.8821s).
23:16:18 INFO [Interpreter.py8849]: external event found: executeProgram.success
23:16:18 INFO [compilor.py8873]: Entering State PSN KRL01
23:16:18 INFO [compilor.py8873]: Entering State Record Angles
```



INT
IS
OMI
STARK

FSMT Facts

- Python implementation
 - Pyscxml: scxml execution
- Handles timing and environment setup
- Component start-up automated
- Not coupled to robotics, but motivated by robotics requirements
- Integrated into Jenkins
- External STRANDS project uses FSMT to test a patrol robot scenario

Working solution for functional system testing

The work on FloBI meets Jenkins and FSMT is supervised by PD Dr.-Ing. Sven Wachsmuth (CITEC) and Dr.-Ing. Ingo Lütkebohle (IPVS University of Stuttgart)





Summary

- Build C++ projects
- Job structure
- Job generator tool
- Installation and continuous development support for distributions
- Functional testing of FloBi with FSMT

Jenkins is used for all aspects and provides the required automation support.



General Jenkins Feedback



- LTS is a good thing
 - In Mainline small things broke frequently with updates
- Web Interface easily becomes slow and annoying to use
 - Some plugins seem to be causes for this
- Scheduling for native code
- Support for multiple LDAP servers

 **Thank You To Our Sponsors**



Platinum



Gold



Silver

