Hillah, Maesano, De Rosa, Maesano, Lettere, Fontanelli

Service Functional Test Automation

UPMC, SEF, DEDALUS S.p.A

Joint STV / INTUITEST Workshop @UCAAT 2015

Background

- Modeling and Verification team in LIP6/UPMC
  - Specification, modeling and verification of distributed systems (SPL, SPEM, test, model checking, SAT / SMT)
- DECISION team in LIP6/UPMC
  - Theory of decision, algorithmic optimisation, OR, AI
- Healthcare Distributed Systems (DEDALUS)
- Services Architectures, test (SEF)

Challenges

- Service Functional Testing Automation is hard
  - end-to-end test of complex, distributed service architectures
  - black-box (services) and grey-box (architectures)
- Configuration of the test execution system
- Constraint-based test input and oracles generation
- Intelligent dynamic scheduling of test cases
- Intelligent reactive planning of test campaigns

Context

- Calabria Cephalalgic Network (headache integrated care processes)
- Multi-owner Services Architecture, Cloud deployment
- APIs HL7/OMG HSSP Standard compliant
- DEDALUS in charge of RLUS, IXS, and CTS2

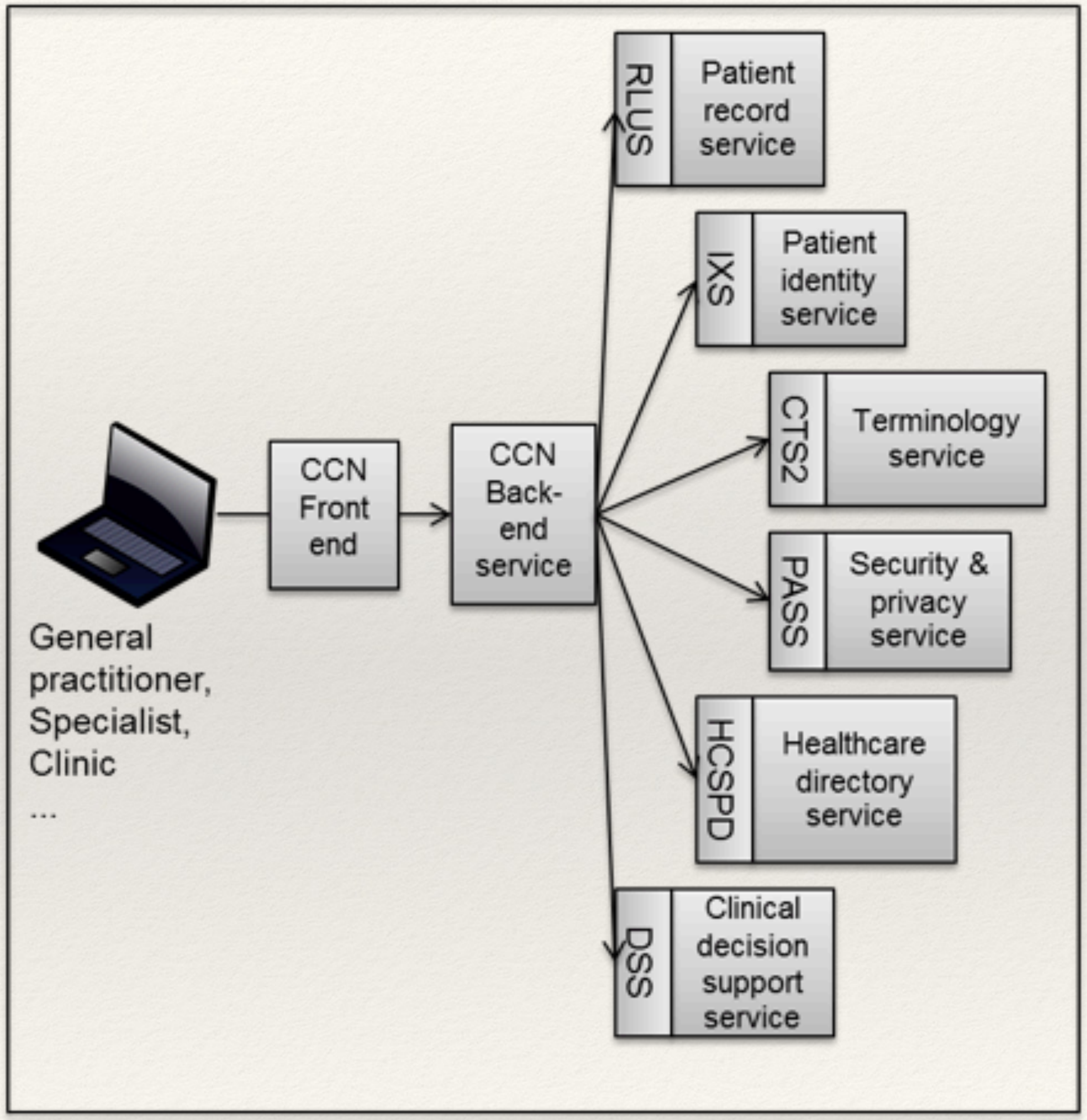

General practitioner, Specialist, Clinic ...
CCN Front end
CCN Back-end service
RLUS
Patient record service
IXS
Patient identity service
CTS2
Terminology service
PASS
Security & privacy service
HCSPD
Healthcare directory service
DSS
Clinical decision support service

Figure 1. Calabria Cephalalgic Network.

Testing Process and Goals

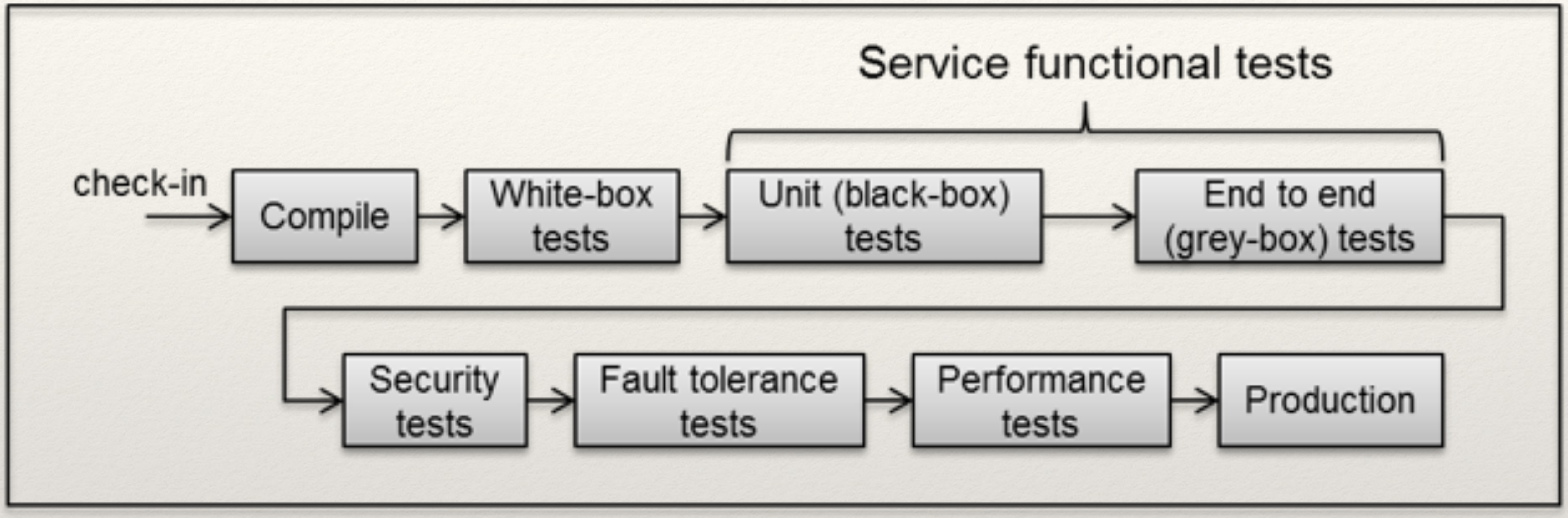


Figure 2. Service integration process as a pipeline.

- Maximise fault exposing potential, fault detection rate, and troubleshooting efficacy
- Improve agility, and time-to-market

The MIDAS Approach for Service Functional Testing

- Through functional test automation, provide a SaaS-based solution for:
  - optimised generation of inputs and oracles
  - optimised management of test suites for first testing, re-testing, regression testing
- **Techniques**:
  - **automated test system configuration and execution**
  - **automated test case generation (inputs/oracles)**
  - **automated scheduling of test execution**
  - **automated reactive planning**

MIDAS Functional Testing Overview

- **From Input Models:**
  - Service model (WSDL, XSD) - prerequisite
  - Service Architecture Under Test (SAUT) model (structural) - topology of components and services
  - Protocol State Machine (PSM) model (behavioral) - behavior at the interfaces
    - business rules (pre/post conds., transfer functions)
    - alternatively to PSMs, Interaction Path Models (e.g. sequence diagram)
- **Generation of Test Suites**
  - Interaction paths with actuals payloads
- **Scheduled execution of test suites**
  - Probabilistic inference for failure searching and troubleshooting
- **Generation of TTCN3 library (executable)**
- **Scheduled execution and on-the-fly generation (planning)**
  - Probabilistic inference also for controlled test generation

Test Environment

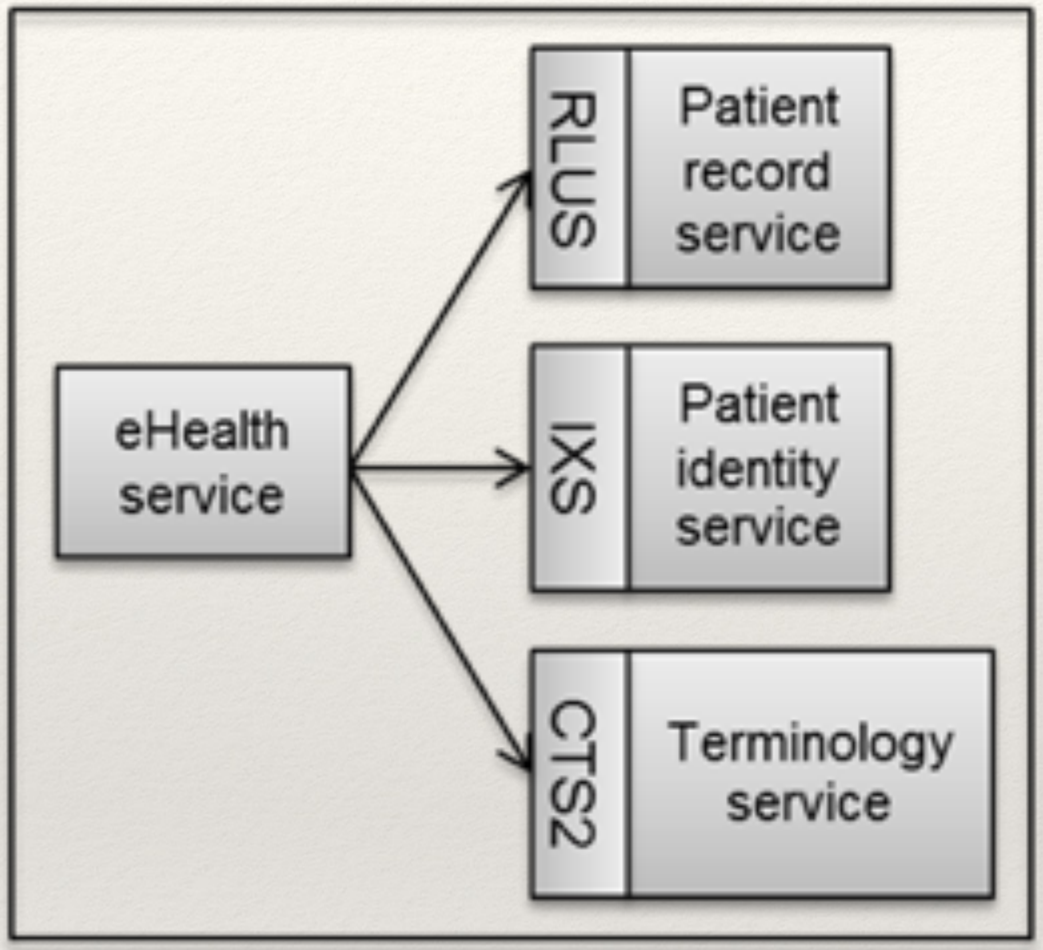


Figure 3. Services architecture under test.

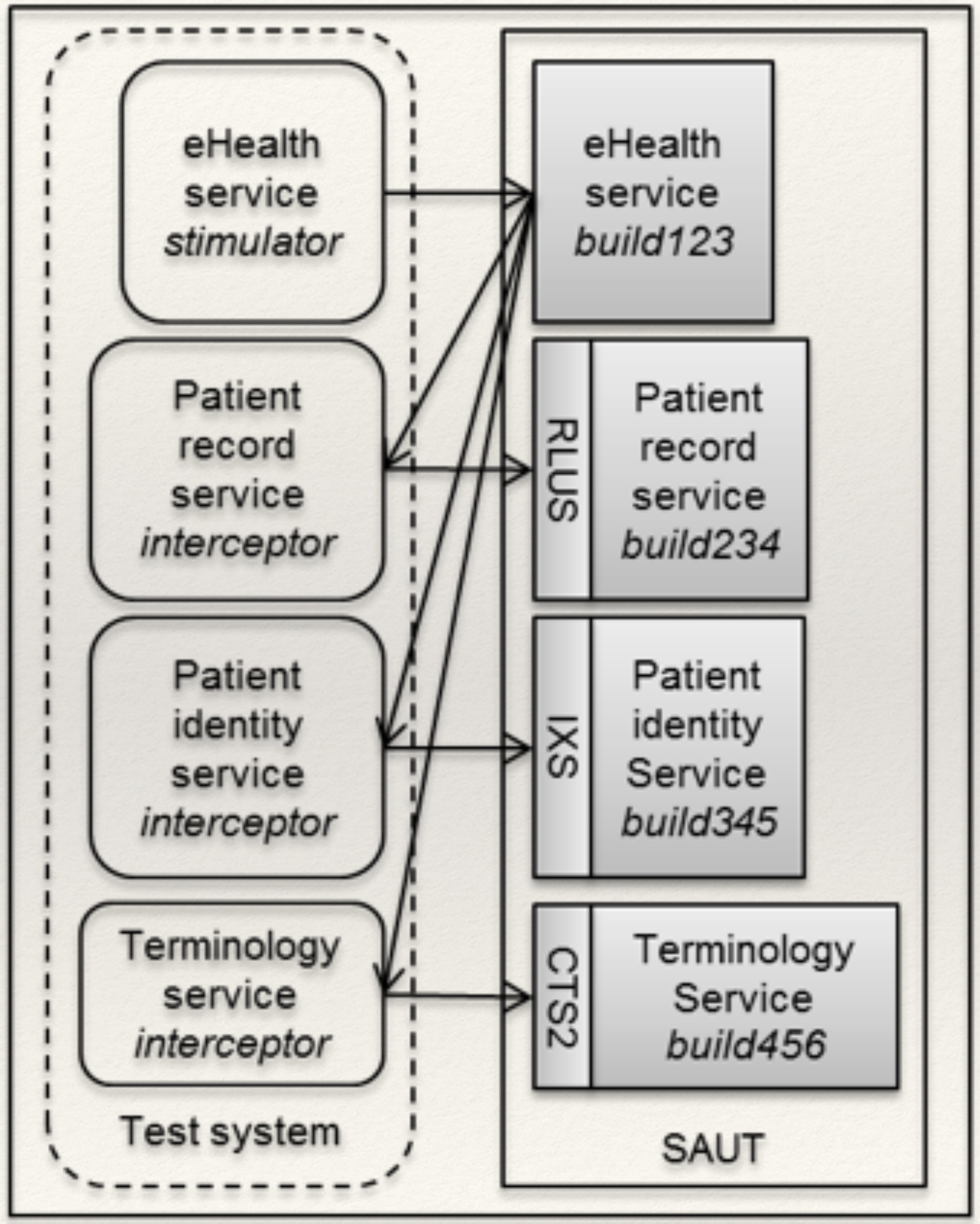

eHealth service *stimulator*
Patient record service *interceptor*
Patient identity service *interceptor*
Terminology service *interceptor*
Test system
eHealth service *build123*
RLUS
Patient record service *build234*
IXS
Patient identity Service *build345*
CTS2
Terminology Service *build456*
SAUT

Figure 6. Test environment for end to end test.

Automated Test System Configuration

- Test system structure: stimulators, mocks, interceptors
  - generated from SAUT and test config. models
- SAUT: Service Components Architecture (SCA) and Service specifications (WSDL)
  - actual components and wires between them
- Test config. model: add virtual components (stimulators, mocks) and virtual wires
  - interceptors for actual wires to be observed

Automated Test Case Generation

- PSM: Standard SCXML documents
- Conditions and transfer functions in Javascript
- Model checking using TLA+ framework for test input generation
- PSM execution for test oracle generation

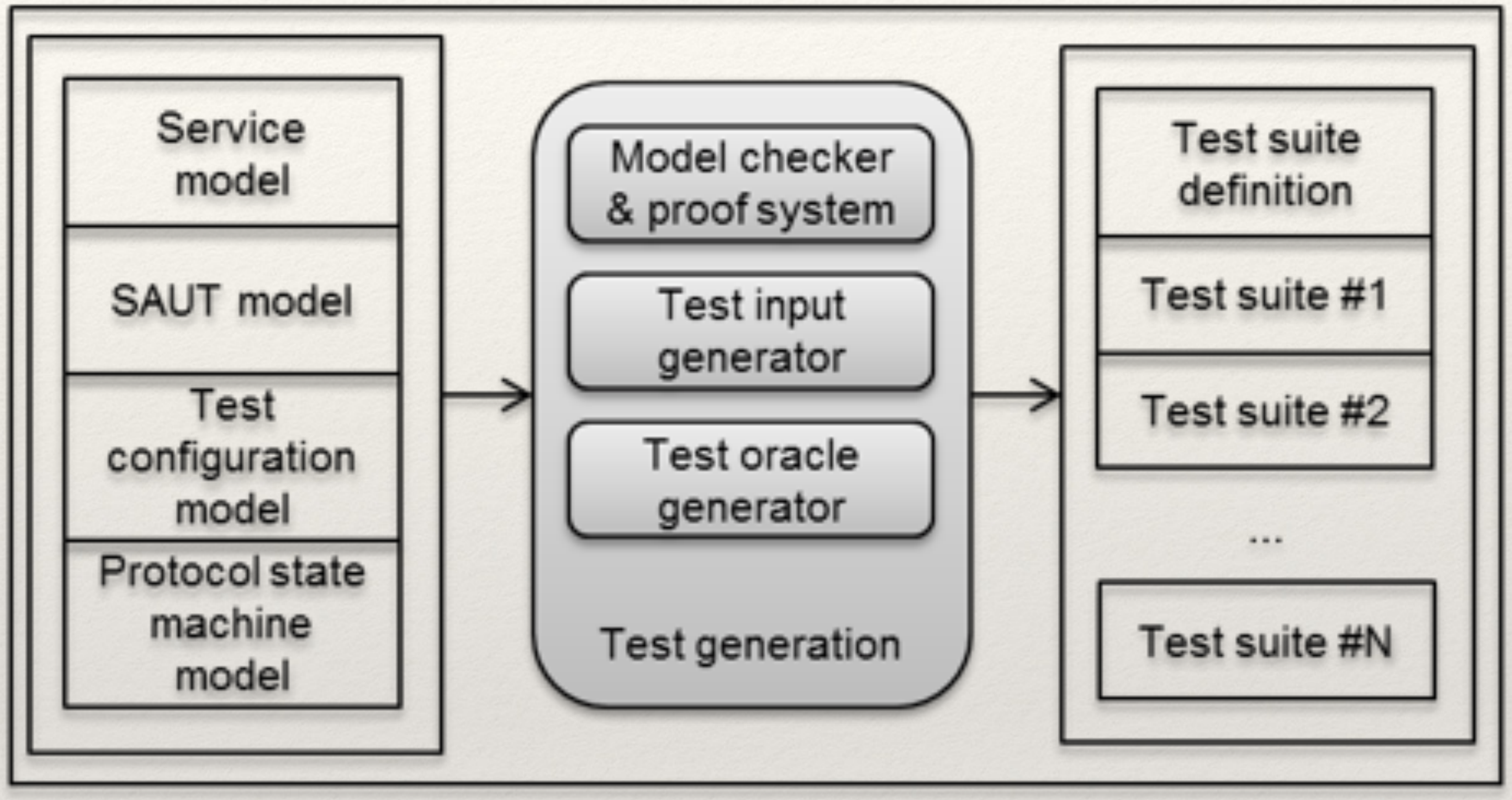
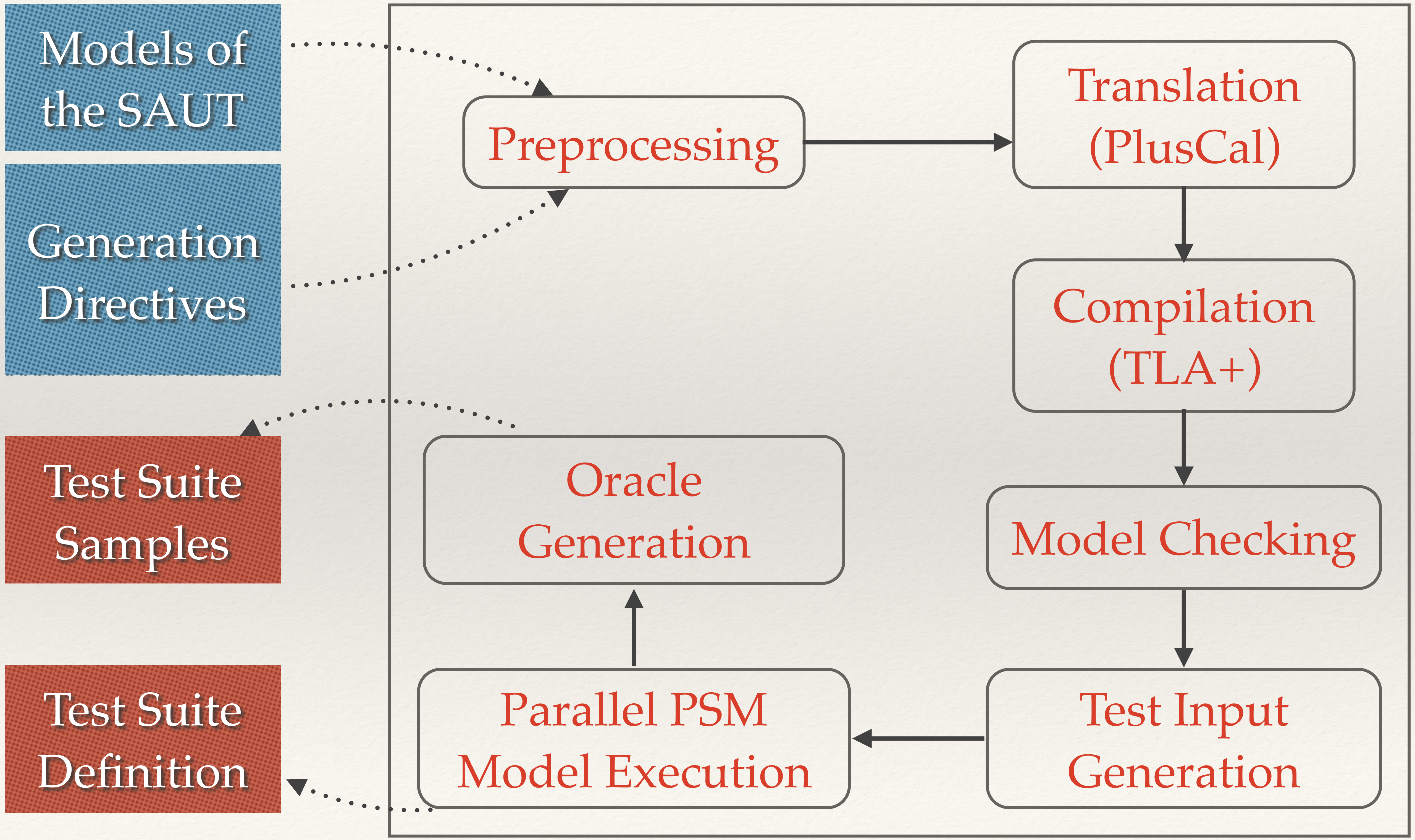


Figure 7. Automated generation of test cases.

Test Case Generation Overview



Automated Test Scheduling

- Cycle *schedule/execute/arbitrate*
- Choose the next test case to run on the basis of past test verdicts
- Detect failures early, and locate faulty elements (troubleshooting)

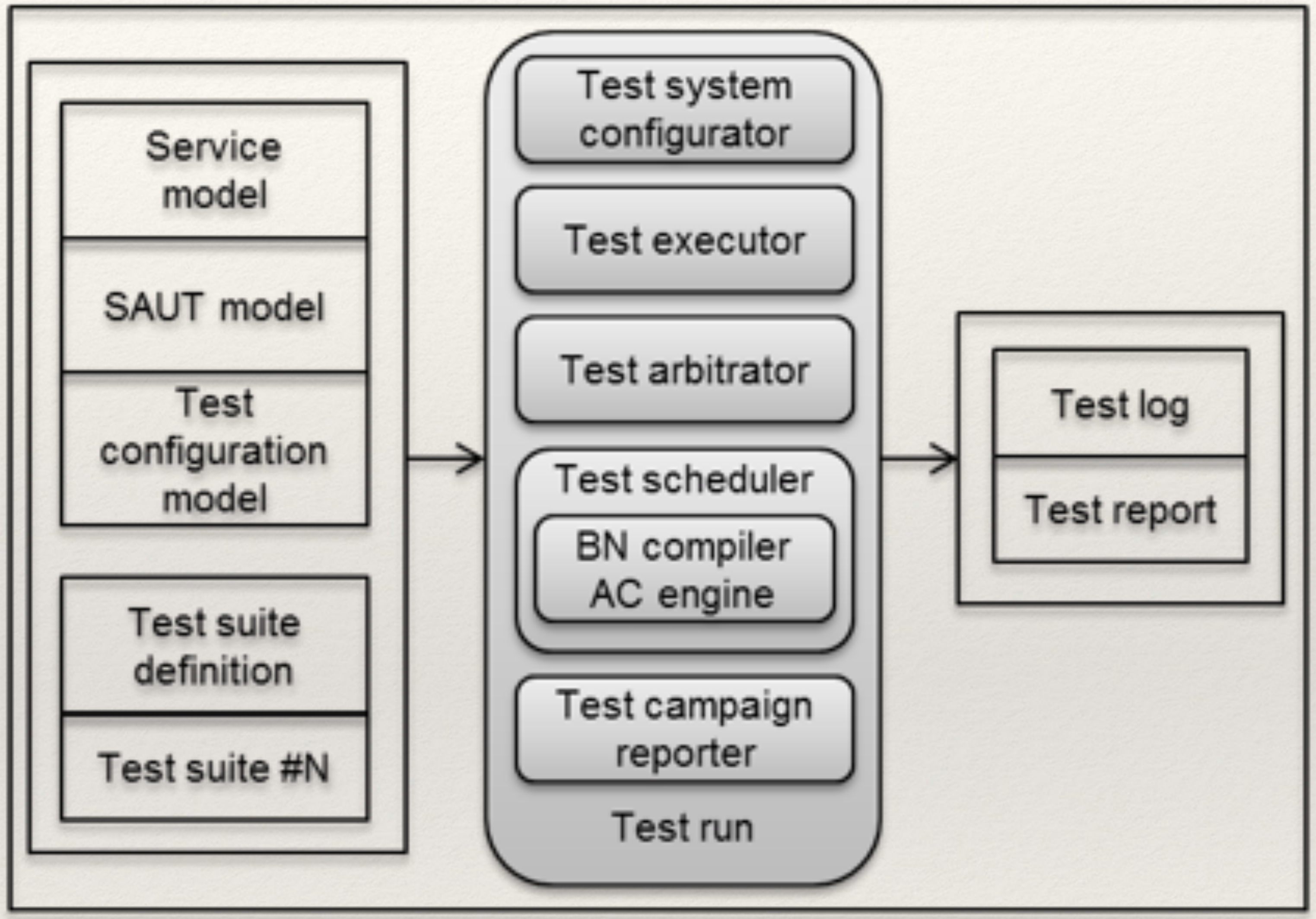
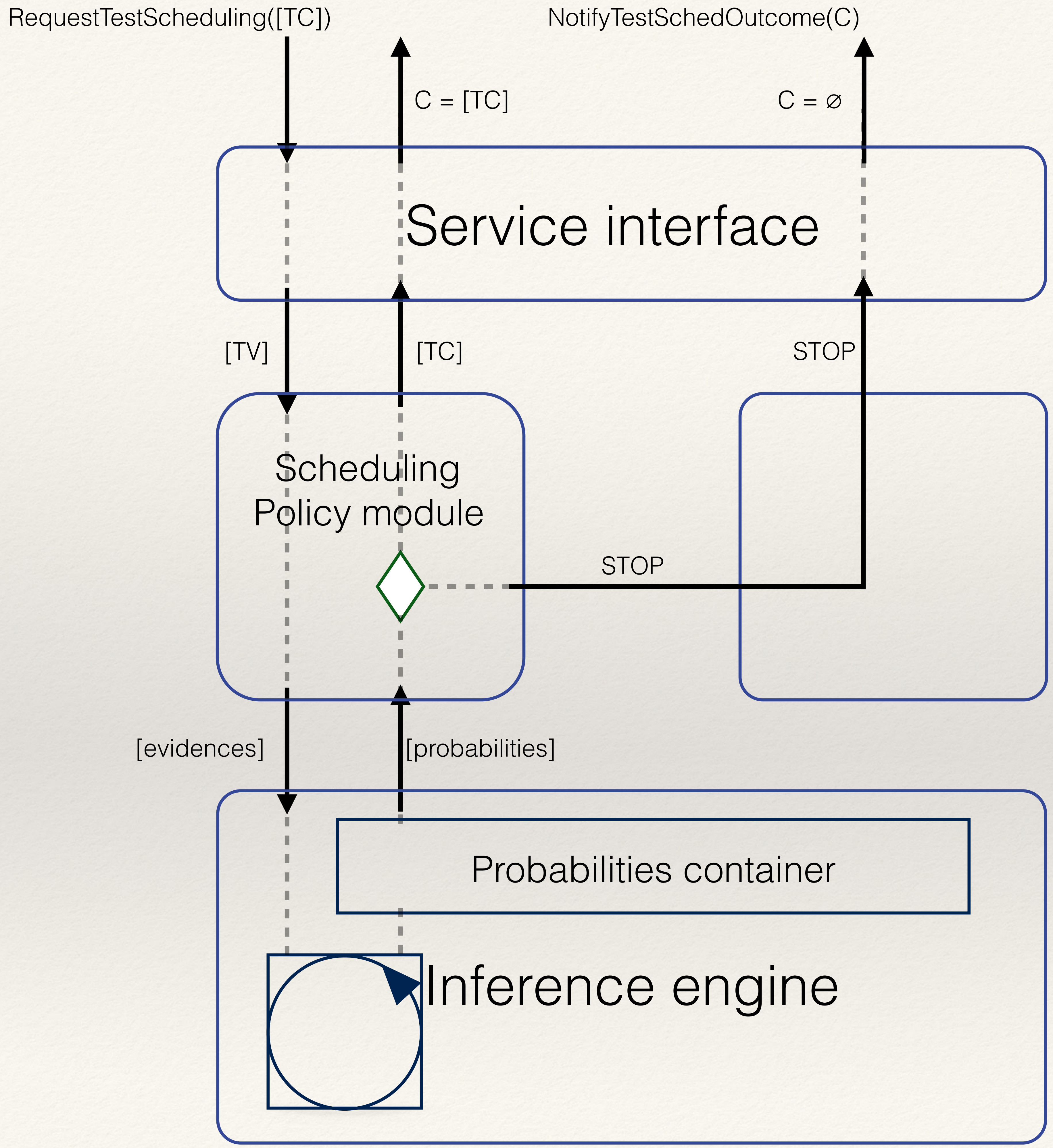


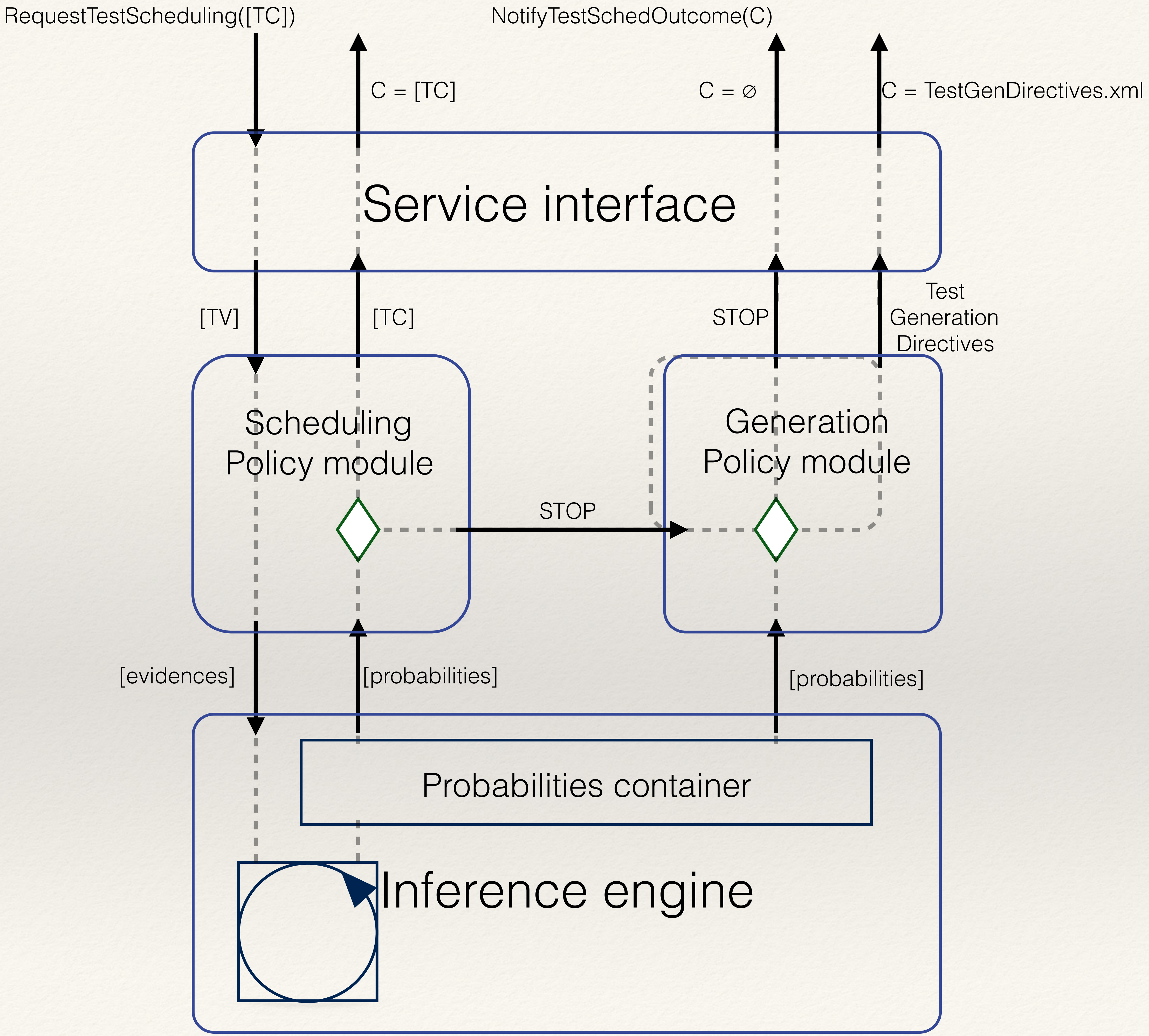
Figure 8. Automated scheduled execution of test cases.

<u>Prioritisation of test cases based on probabilistic reasoning</u>

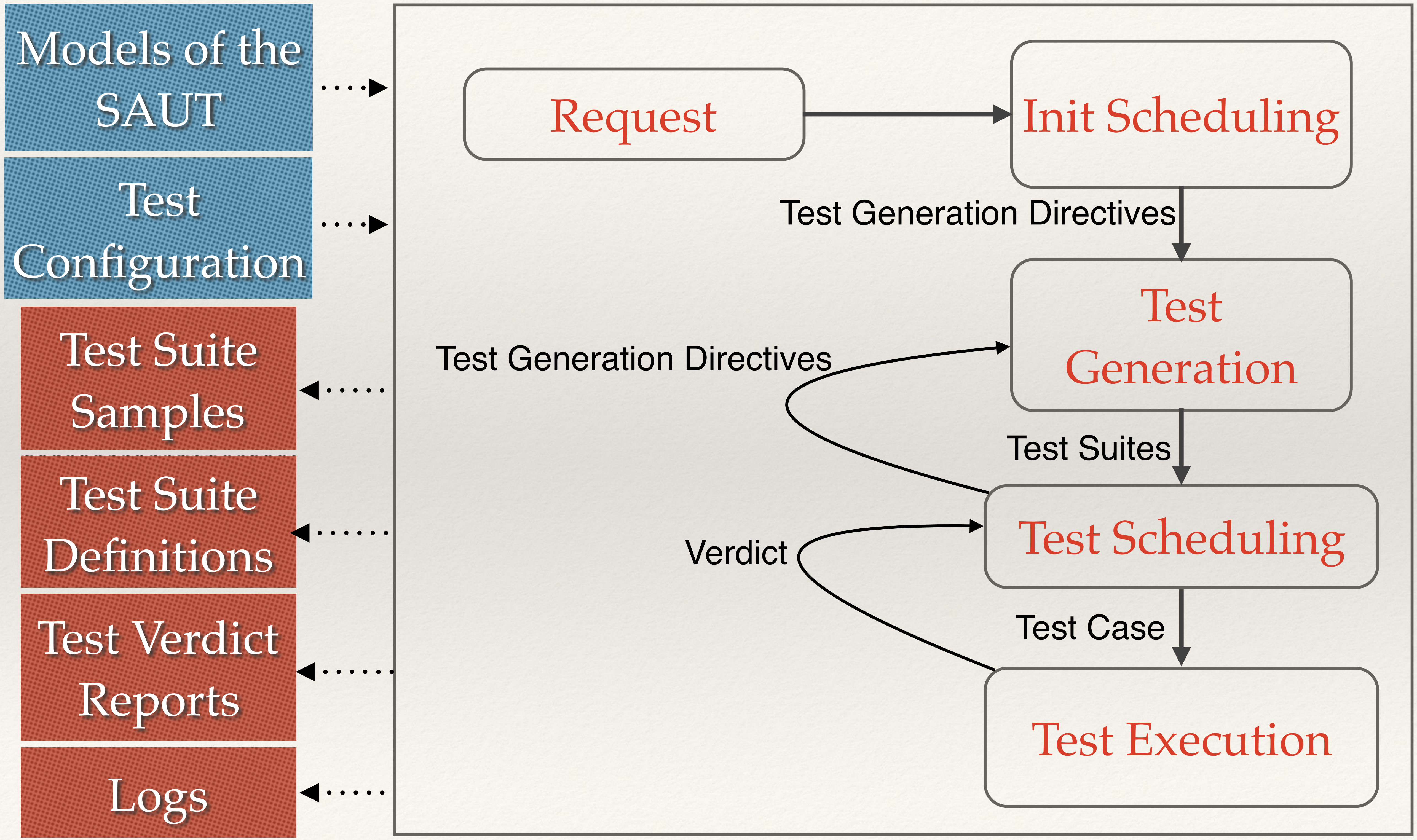

RequestTestScheduling([TC])
NotifyTestSchedOutcome(C)
C = [TC]
C = ∅
Service interface
[TV]
[TC]
STOP
Scheduling Policy module
STOP
[evidences]
[probabilities]
Probabilities container
Inference engine

Automated Reactive Planning

- Scheduler not only drives the choice of the next test case to execute, but also of the on-the-fly generation of new test cases
- Using evidences from past test runs:
  - calculates the degree of ignorance of SAUT elements and recommends the generation of test cases whose execution would diminish this ignorance


RequestTestScheduling([TC])
NotifyTestSchedOutcome(C)
C = [TC]
C = ∅
C = TestGenDirectives.xml
Service interface
[TV]
[TC]
STOP
Test Generation Directives
Scheduling Policy module
Generation Policy module
STOP
[evidences]
[probabilities]
[probabilities]
Probabilities container
Inference engine

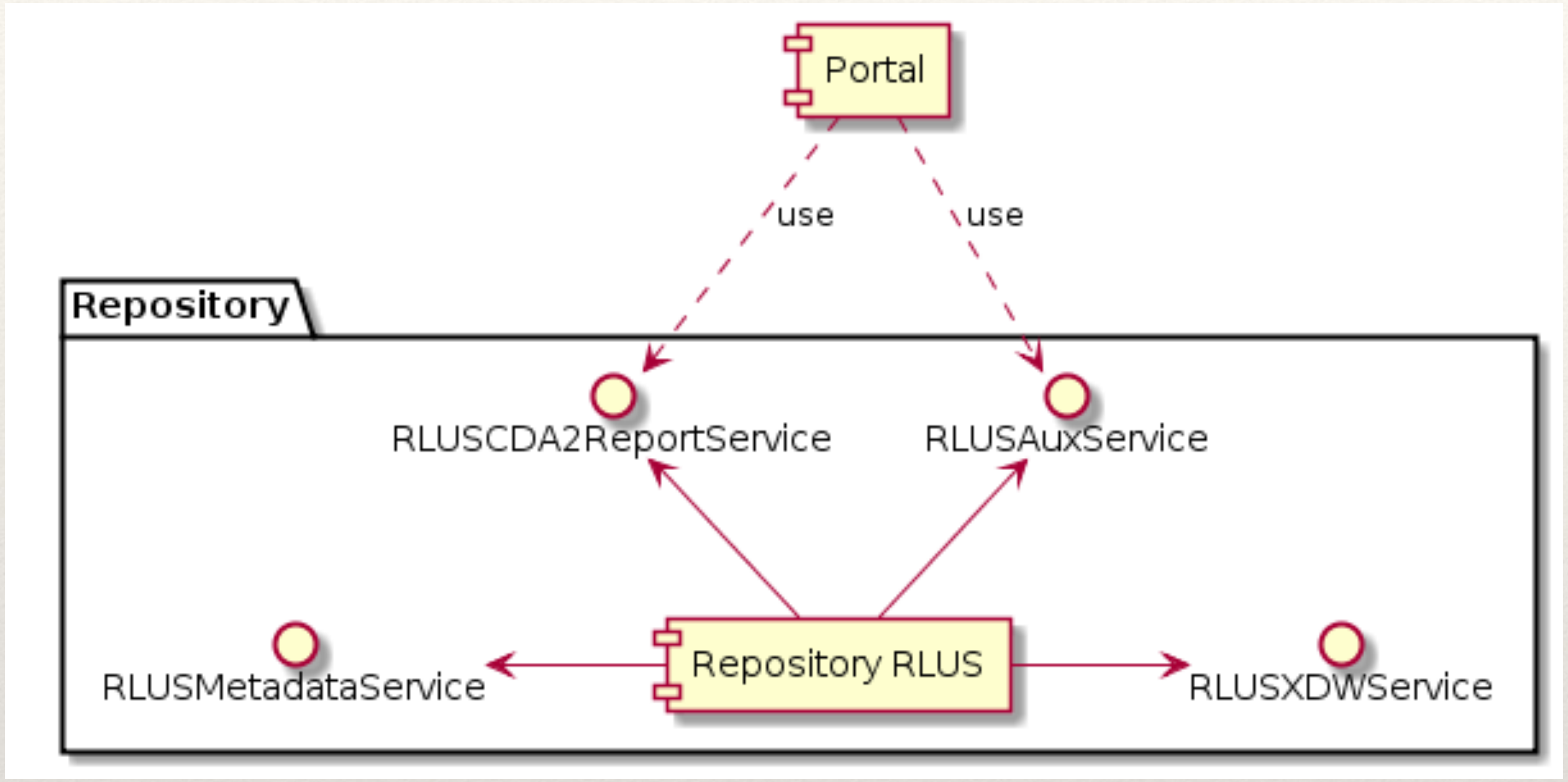
Functional Testing Workflow Overview

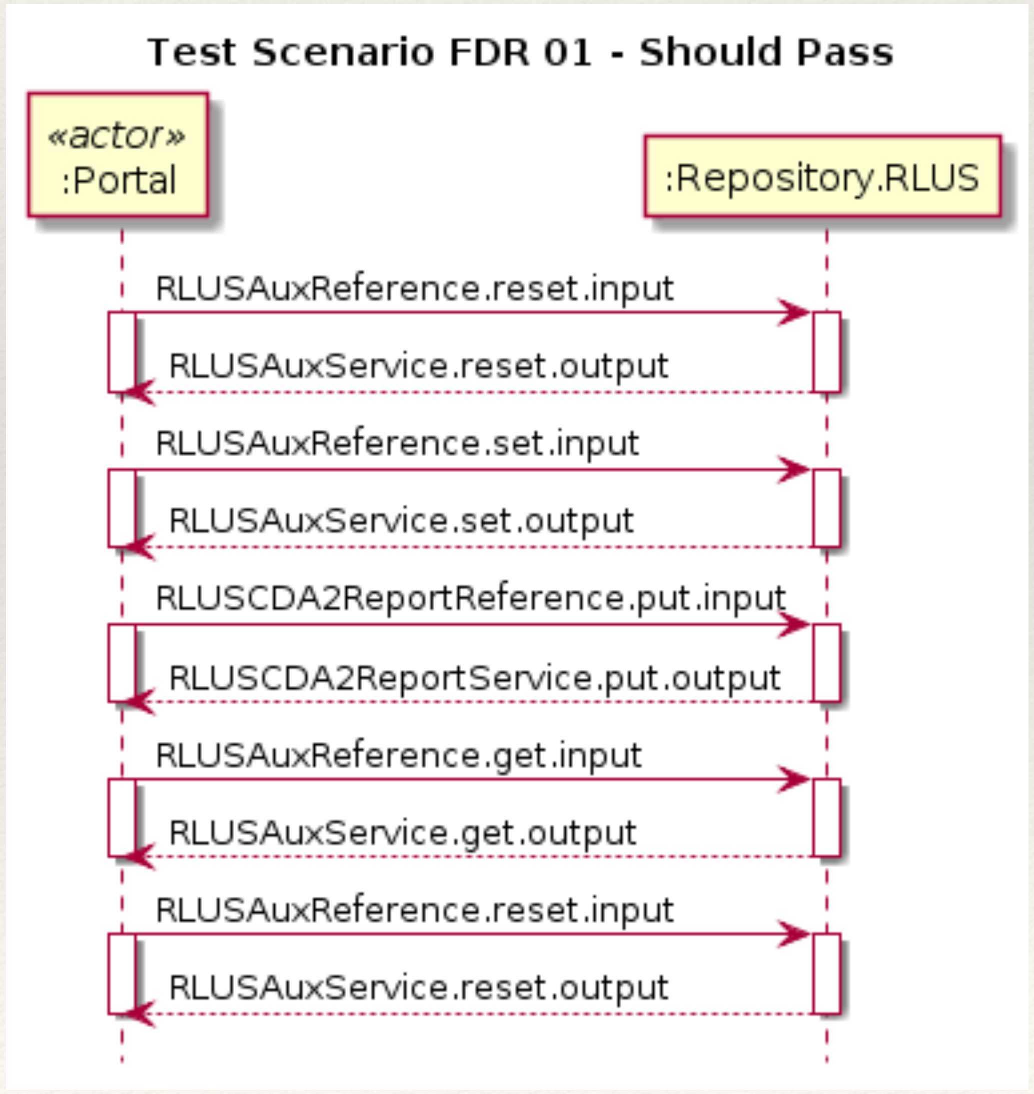


Prototype

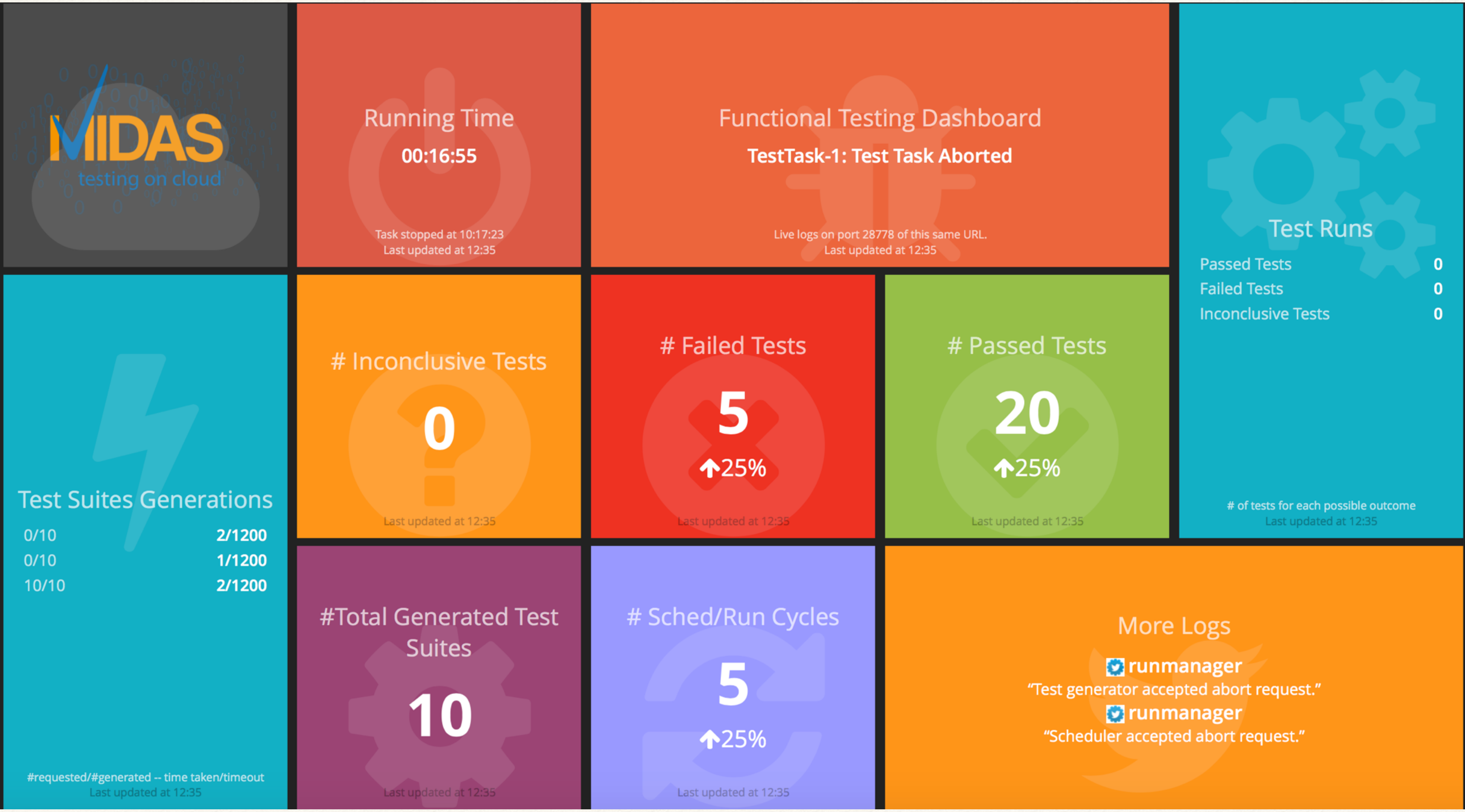
- Test automation methods provided as services
  - can be combined in service integration and delivery processes (continuous integration / delivery)
- Deployed on AWS
- Currently being evaluated by DEDALUS

SAUT - Example




Test Scenario FDR 01 - Should Pass
«actor» :Portal
:Repository.RLUS
RLUSAuxReference.reset.input
RLUSAuxService.reset.output
RLUSAuxReference.set.input
RLUSAuxService.set.output
RLUSCDA2ReportReference.put.input
RLUSCDA2ReportService.put.output
RLUSAuxReference.get.input
RLUSAuxService.get.output
RLUSAuxReference.reset.input
RLUSAuxService.reset.output

Prototype Dashboard



MIDAS testing on cloud

Running Time
00:16:55
Task stopped at 10:17:23
Last updated at 12:35

Functional Testing Dashboard
TestTask-1: Test Task Aborted
Live logs on port 28778 of this same URL.
Last updated at 12:35

Test Runs

Passed Tests	0
Failed Tests	0
Inconclusive Tests	0

# of tests for each possible outcome
Last updated at 12:35

Test Suites Generations

0/10	2/1200
0/10	1/1200
10/10	2/1200

#requested/#generated -- time taken/timeout
Last updated at 12:35

Inconclusive Tests
0
Last updated at 12:35

Failed Tests
5
↑25%
Last updated at 12:35

Passed Tests
20
↑25%
Last updated at 12:35

#Total Generated Test Suites
10
Last updated at 12:35

Sched/Run Cycles
5
↑25%
Last updated at 12:35

More Logs
runmanager
"Test generator accepted abort request."
runmanager
"Scheduler accepted abort request."

Conclusion

- Configuration of test system against distributed services architectures
- Test case generation, using model checking and parallel PSM execution
- Intelligent dynamic test case prioritization and scheduling
- Intelligent reactive planning of test campaign with on-the-fly, evidence-based generation of new test cases

Perspectives

- REST/JSON Service testing; Application to Logistics, IoT
- Automated check of the alignment of the SAUT deployment with the SAUT model
- Test oracles generated from incomplete specifications
- Improvement of test reports for more tester-friendly readability (e.g. trace, diffs, coverage)
- New heuristics for the scheduling (optimised testing strategies)
- Enhance technical evaluation (automated)
- Graphical Modeling IDE for integrated SAUT models (WSDL, SCA, PSM)
    - e.g. XML-based to UTP-based

Q & A

Thank you