Using TTCN-3 to Design Performance Tests

George Din, Cosmin Rentea

Fraunhofer FOKUS, Germany
din.rentea@fokus.fraunhofer.de

Outline

Motivation
Challenges in using TTCN-3 for Performance Testing
Design of Performance Tests
Adaptation Layer
Test Distribution and Load Balancing
Performance Evaluation
Extensions
Conclusions
Motivation

- Standardized Test Technology (ETSI)
- Mature and stable concepts
- Programmable, flexible
  ... but implementation independent
- Execution of parallel and distributed behaviors
- Full test execution control
- Parameterizable
- Reusability of
  - functional tests
  - configurations
  - design patterns

Efficient execution depends on tools and patterns

Challenges

- Test Design
  - data representation
  - how to define behaviors
  - how to emulate big user populations
  - write compact and performant tests

- Adaptation
  - handle a large number of connections to SuT

- Test Distribution
  - the traffic needed for load tests cannot be produced by only one tester
    • benefit of all the computational resources allotted for testing purposes
Design of Performance Tests

```tcl
type component ClientEmulator {
    var ClientData client [NumberOfUsers];
    port PortToSUT clientPort [NumberOfUsers];
}
function ClientBehaviour runs on ClientEmulator {
    ... 
    timer t := 0.5; t.start;
    p[i].send(stimuli) {
        // update state of user i
    }
    any port.receive(response) {
        // identify user and go to next state
    }
    altstep Error runs on ClientEmulator {
        any port.receive(unexpectedResponse) { ...
    }
}
```

• define **workload units**
• handle multiple ports for the connection to SUT
• keep internal data about emulated users
• different types of users at the same time

• emulate **concurrent and dynamic user behaviors**
• use timers for responsiveness evaluation
• use **send/receive** paradigm
• use **templates** for stimuli and response validation

• **exception handling**
• access internal data of the component where the altstep is running

---

**TimedTTCN-3 (2001)**

- **Test Non-functional Conf verdict**
  - pass -> **conf** -> inconc -> fail -> none

- **Logfile**
  - local to each test component
  - first, next, previous, retrieve

- **Local clock**
  - now, resume, read, wait

- **Timezones**
  - specification of clock-synchronized test components
Data Representation

Data interchanged with the SUT has to be:

1. defined (TTCN-3 types)
2. instantiated (TTCN-3 templates)
3. encoded (TCI, adaptation layer)
4. decoded (TCI, adaptation layer)
5. matched (template matching)
6. processed (TTCN-3 code)

Behavior Definition

- Population = the number of users emulated during an execution of a performance test

- Aware of:
  - a test component is generally a thread (or process) in the generated code
    - 1000 x test components => 1000 x threads
  - alt blocks with alternatives
Main Goal - High Parallelism

- For most performance tests we need to run simultaneously as many actions as possible.

- Behavior = Ts \rightarrow S \rightarrow W \rightarrow R \rightarrow Tr

  - B1 = Ts \rightarrow S \rightarrow W \rightarrow R \rightarrow Tr
  - B2 = Ts \rightarrow S \rightarrow W \rightarrow R \rightarrow Tr
  - B3 = Ts \rightarrow S \rightarrow W \rightarrow R \rightarrow Tr
  - B4 = Ts \rightarrow S \rightarrow W \rightarrow R \rightarrow Tr

send
Response timer

Ts = preparation of the send template
S = send action
W = wait time
R = the response is received
Tr – response template validation

Test Specification Patterns

- One client per component
  - a component stops when a user finishes its behavior
  - easy to apply and distribute

- Sequential repetition of users per component
  - reuse a component for another set of data
  - a component works until the end of the test

- Interleaved user behaviors per component
  - many users are simulated in parallel on a single component
  - highest degree of parallelism
Processing of Messages

Adaptation Layer

**Aim:** Minimize TTCN-3 Runtime processing overhead!

- Handles multiple parallel connections with SUT
- Uses Pools of Threads and Listeners
- Smaller number of
  - threads and context switches
  - mutexes and semaphores
  - contended message queues
- Assures milliseconds precision
Test Distribution

- Test distribution has the goal of selecting a home for each created test component depending on distribution parameters.

- Decision criteria
  - External parameters
    - environment specific: bandwidth, CPU, memory
  - Internal parameters
    - test specific: component type, behavior type, communication pattern

Implementation Architecture

Diagram showing the architecture with components such as Test Console, Session Manager, Load Balancer, and communication middleware.
Test Distribution Language

- **Purpose**: an XML-based language describing how the components are distributed among test nodes using common strategies.

```
<component_assembly>
  <description>Example to use TCDL language</description>
  <special_container="container1"/>
  <set>
    <component_selectors>
      <componenttype>ptcType</componenttype>
    </component_selectors>
    <homes distribution="round-robin">  
      <container id="container1">
        <max_components>10</max_components>
      </container>
      <container id="container2">
        <max_components>100</max_components>
      </container>
    </homes>
  </set>
</component_assembly>
```

Component Assembly

- Define a set of components
- Filtering rules for the components
- List of locations for the filtered components
- Distribution algorithm
- Definition of local constraints for a location
Experiment

- **Hardware configuration**
  - TestNode1 (memory 512 Mb, CPU 1.9 GHz)
  - TestNode2 (memory 2 Gb, CPU 2 x 3.5 GHz)
  - TestNode3 (memory 1 Gb, CPU 3.5 GHz)

- **Test run procedure**
  - increase the number of components per host
  - measure processing time between receiving a message from SUT and sending a new stimulus
  - note: processing time increases with the number of components deployed on the same host

- **Evaluation criterion**
  - the best algorithm is the one for which the processing time is the smallest one

Round Robin vs. Memory Factor

![Graph showing comparison between Round Robin and Memory Factor](image-url)
Performance Evaluation

- Measure overall performance parameters of SUT
  - calls attempts per second
  - successful call establishment rate
  - call establishment delay

- Test Logging Interface (TLI)
  - XML logging of messages
    - interoperable, but verbose
  - processing using SAX events or Antlr-based compilers
    - could be done realtime
  - statistical analysis and graphs using open-source tools
  - complex reports (XHtml, Latex, Pdf)

Traffic Generation

- Define predictable streams of packets following a given stochastic pattern of transmissions times
  - e.g. Poisson, Erlang, normal, hyper-exponential etc.

- Built-in traffic models
  ```java
  Poisson pr = Poisson.create(params);
  myPort.poisson(pr, msgTemplate);
  ```

- Real-time execution environment
  - Operating system
  - Real-time JVM (JSR 1)
Message Arrival Distribution

- **Ptc1**
- **Ptc2**
- **PtcN**

Controller enforces a Poisson distribution

Message Queue

- queue length
- interval computed using distribution

Synchronization

- **Load controller coordination**
  - all components wait for a synchronization event (a special "start" token)
  - controller component - is responsible for simultaneously signalling all waiting components

- **Barrier (like in MPI)**
  
  ```java
  Barrier b = Barrier.create(NumberOfPTCs);
  b.wait();
  ```

- **NTP LAN clock synchronization for all the test nodes**
Statistical Verdicts

• A verdict in a performance test has rather a statistical meaning than a functional one

• Verdict should be established by counting the rate of fails during one execution

• Example:
  – setverdict(pass);
  – setverdict(pass);
  – setverdict(fail);

> the verdict will be: 66.66 % Pass

Conclusions

• Goal: Enabling TTCN-3 to execute performance tests

• Applied Methods:
  – Smart TTCN-3 specifications
    • follow de-facto patterns
  – Use efficient adaptation layers
    • fast, customized protocol stacks and codecs
    • optimized Runtime
  – Load distribution driven by distribution algorithms
  – Exhaustive analysis of test results