

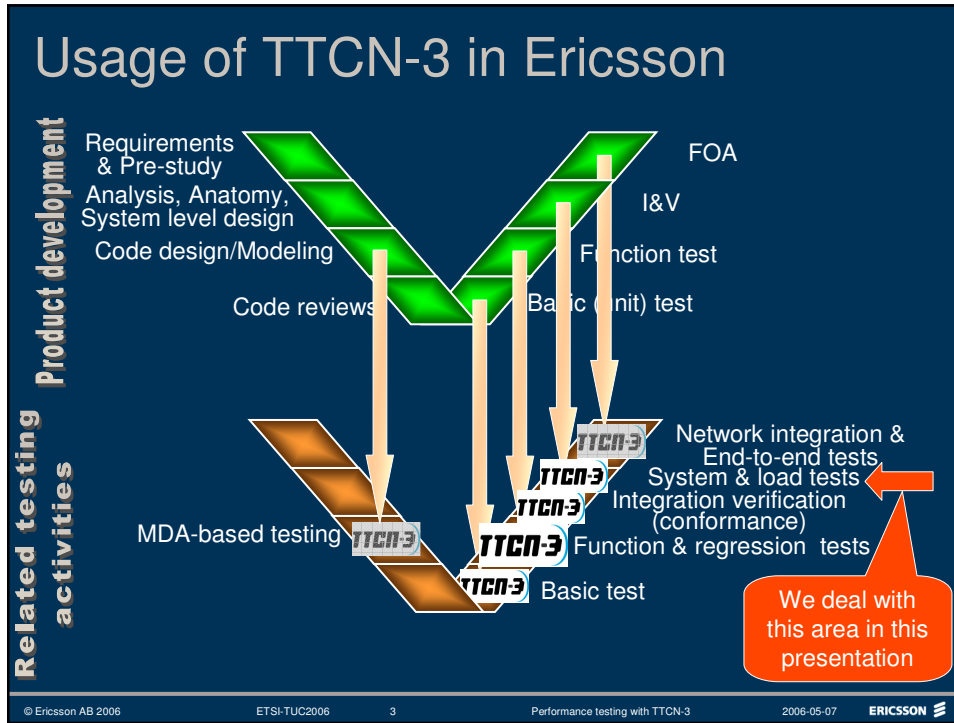
# Performance testing with TTCN-3

Gábor Ziegler and György Réthy  
Ericsson Test Competence Center  
Ericsson Hungary Ltd.  
Research & Development  
{gabor.ziegler,gyorgy.rethy}@ericsson.com



## Contents

- Background
- Why a load framework
- Cornerstones
- Conclusions



- ## Previous TTCN-3 load testing projects
- 2002
    - Research project – HTTP load generation with TTCN-3
  - 2003
    - Radius & Diameter load testing of AAA/EWAS/HSS nodes
    - Beyond 3G research project – emulation, conformance, interoperability and load/performance/stability testing of different IP mobility solutions
  - 2004
    - (Diameter and Radius load tests continued)
    - Load, characteristics, stability and robustness tests of the Automatic Device Configuration solution (TTCN-3 is emulating HLR, BSC, SGSN & SMS-C nodes)
  - 2005
    - Automatic Device Configuration contd., Ericsson Multi Activation 3G charging and mobile positioning load tests
    - ISUP load generator (and terminator)
- © Ericsson AB 2006    ETSI-TUC2006    4    Performance testing with TTCN-3    2006-05-07    ERICSSON

## Experiences from previous projects

- Quotes from users
  - 👉 “Extremely flexible tool that generates traffic load efficiently.”
  - 👉 “Cheap, commercial PCs used as traffic generators.”
  - 👉 “Facilitates automatic testing.”
  - 👉 “Source code available for internal updates.”
    - “Deep programming skills are required for testers.”
    - “Too much time spent on test case writing.”
    - “When used in load mode for high load, we can’t have full control of the load generated.”
    - “We have the feeling that we are not taking advantage of 100 % of TTCN capabilities.”
- There are a lots of common tasks, i.e. a lots of duplication of work
  - be patient, see details on next slides

## Contents

- Background
- Why a load framework
- Cornerstones
- Conclusions

## Load framework: Targets

- To provide an \*easy to use\* solution!
- Many tasks are generic and common, which shall be done only \*once and properly\*, such as...
  - ...ensure exact scheduling of the generated traffic
  - ...provide accurate mix traffics
  - ...manage shared resource pools
  - ...provide run-time MMI interface
    - Start/Stop of load generation
    - On-line, centralized adjusting of parameters
    - On-line, centralized “at a glance” overview of status information
- Increase efficiency, decrease cost via code re-use
  - Many of the products use the same protocols
  - Elementary protocol behaviours are not changing (standardized)

## Load framework: Targets (contd.)

- The TTCN-3 code runs over an “abstract TTCN-3 machine”
  - How to optimize an abstract language?
  - Code optimization should take into account the peculiarities of our TTCN-3 executor
    - Insider knowledge from the tool developer is needed
- To provide guidance to TTCN-3 users
  - Test configurations/architectures for load testing
  - Design rules for test designers
  - \*Howto\*s: advises to increase execution performance

## Phases

- Study phase
  - From October 2005 till March 2006
  - Collect and consolidate user requirements
  - Develop an implementation sketch and packages
  - Provide first howto guideline
- Implementation phase
  - From April 2006

## Contents

- Background
- Why a load framework
- Cornerstones
- Conclusions

Most cited requirement:

It must be  
easy to use!

## Requirements identified

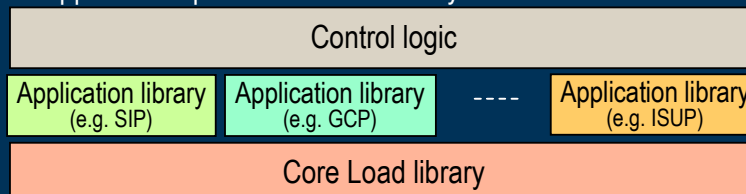
- Requirements are classified into groups of
  - Functional requirements
  - Generic simulation entities / classes support
  - Load generation requirements
    - scheduling
    - scalability
    - routing / broking
    - error handling
  - Logging / tracing / monitoring requirements
  - Detailedness of simulation
  - Statistics requirements
  - General requirements
  - Absolute Numbers / Projects specific requirements
  - Documentation requirements
  - Efficiency requirements

## Three cornerstones of the solution

1. TTCN-3 software framework,
  - A software library
    - The core library as an individual product.
    - Each application dependent libraries are individual products
  - Well defined and documented API
    - Both for the core and the application libraries
  - A collection of *mandatory "Design Rules"* (DR-s) for the testers
    - DR-s for *source-code structure specification*
    - DR-s for *test component structure* (i.e., TTCN-3 PTC hierarchy)
    - *Application specific DR-s*
    - *Best-Practice Guides* for designing effective TTCN-3 code (e.g., use inout parameters instead of a 'return' clause, etc.)
2. Improvements and additions to the Ericsson TTCN-3 Toolset (TITAN)
  - E.g. implementing component extensibility
  - Runtime GUI and CLI
3. Amendments to the TTCN-3 language
  - Though lots have been done to support load testing in edition 3 (e.g. alive PTCs), some features are still missing (see later)

## SW library dilemma: How to optimize?

- Two contradicting requirements:
  - A framework shall be general → generalization
  - Optimization always task specific → specialization
- The solution: Provide a three levels approach
  - "Core framework library": code provided centrally
    - Run-time (on-line) configuration settings,
    - Run-time (on-line) statistics handling
    - Scalability support
    - Generic issues, such as provide pool-variable API
  - "Application specific framework libraries": code provided in-cooperation with project experts, relies on core library code
  - "Control logic": is provided by the "simple tester", relies on application specific and core library code

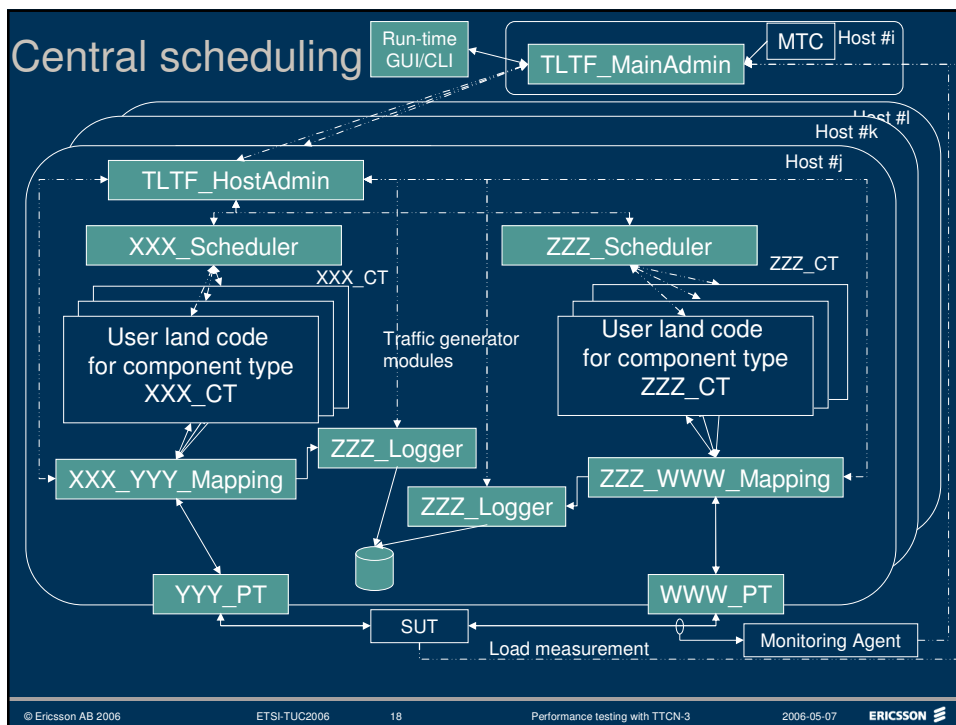
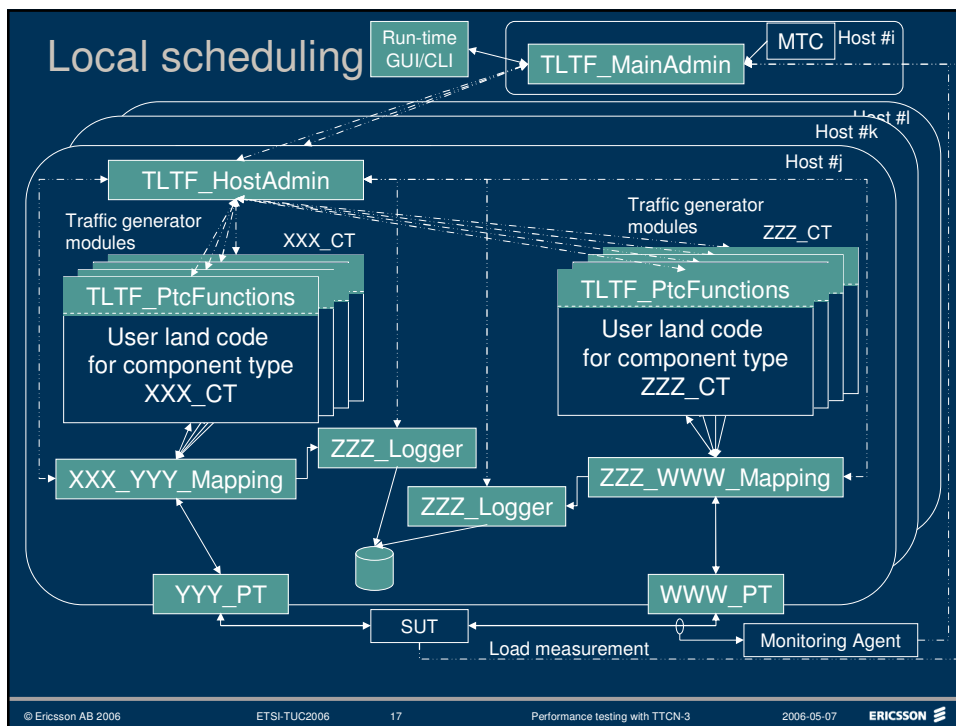


## Shortages of the TTCN-3 language

- Function references (altstep ~, testcase ~)
  - Handling of huge mass of subscribers/context/etc. is problematic today: each is in different state, impossible to provide generic algorithms
  - Solution: support callback functions
    - allow to “register” functions (e.g. store references in a component variable array or pass in as parameter)
    - the generic algorithm will “call-back” the right function whatever the actual state of the entity is
- Non-mandatory parameters
  - Backward-compatible extension of libraries is problematic today: adding new parameters to existing interfaces spoils “old” calls
  - Solution: backward compatible extension of formal parameter lists
    - allow to add default values to formal parameters and
    - allow omitting them in actual parameter lists (default value is used)
- Some object orientation features
  - Libraries normally have
    - public functions & altsteps and
    - auxiliary data, functions & altsteps
  - The latter is subject to change at library upgrades, hence should be made hidden from the user

## Load testing architecture

- Problem: *concurrency handling*
  - Guaranteed failure for load/stress testing: to be polite
    - Initiating the next traffic flow on schedule shall be done independently from the success/failure of the previous tasks
  - Remember: load testing means *parallel* traffic flows over common shared resource pools → concurrency!
  - TTCN-3 has a special concurrency model
    - PTC-s are run concurrently
    - A PTC is a “single CPU system”
      - No concurrency support below PTC level by the language
    - PTC-s are run isolated from each other (no shared memory)
- Solution:
  - Two alternatives, see scheduling strategies on next slides



## Conclusions

- TTCN-3 language and the Ericsson TTCN-3 toolset (TITAN) are mature for load testing,...
- ... however, there are some problems for load generation
  - The abstract nature of the language
  - The special concurrency model of the language
  - Code optimization is a must
    - But the average human “tester” is usually not a competent software designer
- The ongoing Load Test Framework project remedy these shortcomings
  - Provides ready made solution for common problems across projects
  - Provides tool support for runtime control of TTCN-3 test suites
  - Provides Howtos, Design Rules and examples (templates) for application specific problems

Thank you for your attention!  
Questions?

