Regression Testing of DDS-based Systems

“user-driven requirements for a ‘Tester’ tool”

J.H. van ‘t Hag
OpenSplice Product Manager
PrismTech
Hans.vanthag@prismtech.com
Introduction

☐ This presentation is about:
  ☐ **Experiences of system-integrators** in the large-scale naval combat systems domain
    ☐ Focussing on their needs w.r.t. regression testing
  ☐ Separating **requirements** targeting:
    ☐ Testing of individual applications
    ☐ Integration of (3rd party) subsystems into the mission system
    ☐ Analyzing system-wide data-flows
    ☐ Capturing data and convert/use into repeatable test-scenario’s
  ☐ **Explaining concepts** using ‘their’ evolved tool-suite
    ☐ Developed ‘by testers for testers’
    ☐ As a complementary tool to available white-box DDS-tooling (OpenSplice Tuner)
    ☐ Based on the (open-source) Control &Monitoring API of OpenSplice DDS
    ☐ Exploiting the Java-platform and related open-source plugins
  ☐ Recently enhanced with a ‘system browser’ and now productized by PrismTech
    ☐ Available as “**OpenSplice Tester**”, part of OpenSplice V6.1
DDS systems
“a perfect match for testing”

- Autonomous components
- Interacting only with the information they exchange
- Dynamic discovery of (late joining) applications
- Standardized meta-data capturing discovered DDS-entities
- A ‘natural fit’ for a dynamic (regression-) test tool to ‘stimulate’ the system
1. BACKGROUND
1.1 Naval CMS - example

SMART-L
APAR
KH/SCOUT NAV. RADAR
HELI APPR. RADAR
VESTA
OFF LINE R/S
SIRIUS
TDS (2x)
POS
SYSTEM
SHIP'S REF.
IRC

VIDEO INTERFACE CABINET
LG. SCREEN DISPLAY 1
DEPARTMENT OFF. / OFF. QUARTERS
WORKSTATION 01
COTS & SIGMA TYPE PROCESSOR
TORPEDO WEAPON SYSTEM
MK32 MOD-9
(MK46)

INTEGRATED MACHINERY CONTROL SYSTEM
COMMS NETWORK
ATAS
SPHERION + XBT / XSV
+ SURF. TEMP. REC.
ATAS
SPHERION + XBT / XSV
+ SURF. TEMP. REC.

GUN 127 MM
HARPOON

30 MM GUNS OFF LINE
MK41 SM2 LAUNCHER
GOALKEEPER

CATV BUS
LINK 11
TOLPP
VESTA
VESTA
SRBoc

LARGE SCREEN DISPLAY 2
STAFF ROOM
BRIDGE
COMMS ROOM

VIDEO INTERFACE CABINET
LG. SCREEN DISPLAY 1
DEPARTMENT OFF. / OFF. QUARTERS
WORKSTATION 01
COTS & SIGMA TYPE PROCESSOR
TORPEDO WEAPON SYSTEM
MK32 MOD-9
(MK46)
1.2 Large-Scale Naval CMS

☐ >100 Application Developers (C/Java)
  □ Developing data-centric ‘autonomous’ DDS applications
  □ HMI, generic business-logic, embedded Sensor/Weapon interfaces
  □ Need tools for debugging, tuning and testing (individual) applications

☐ Many 3rd party Sensor/Actuator subsystems
  □ Repeatable Subsystem (interface-) testing/validation is essential
  □ Both ‘standalone’ as well as when integrated in the total system
  □ Need tools to test & verify (validate) correct subsystem interfacing & behaviour

☐ All integrated into a single large-scale mission-critical DDS system
  □ 2,200 of applications distributed over hundreds of computers
  □ Up to 150 applications on a single node (6,000 writers, 2,000 readers per node)
  □ Need tools to explore/understand, stimulate/analyze, test/re-test
1.3 Testing Needs

☐ Application developers
  ☐ Want Whitebox debugging/tuning i.e. looking ‘inside’ an application, inspect data, modify QoS, ..
  ☐ Want Blackbox testing i.e. stimulate inputs, monitor outputs, analyze behavior
  ☐ Want an integrated approach to white-box and black-box testing

☐ Subsystem integrators
  ☐ Want a intuitive / easy “language” to express DDS-based interactions with subsystems
  ☐ Want to execute such scripted interactions in batch-mode to automate regression-testing

☐ System testers
  ☐ Want to browse the distributed system from logical and/or physical perspective
  ☐ Want to observe and capture DDS-based interactions between system components
  ☐ Want aids to convert these observations in repeatable scenario’s (scripts) to test the system

☐ … and lets not forget.. Middleware developers (like us)
  ☐ Want to perform performance regression-testing of OpenSplice DDS itself (as part of QA process)
  ☐ Using the regression-test tool to ‘drive’ the open-source DDS Touchstone benchmarking suite
2. “Tester” Architecture
2.1 Major components

☐ Usage aspects concentrated in ‘modules’:
  □ (remote) DDS **connection** module
    □ Allows the tool to ‘interact’ with the DDS-system
  □ System **browser** module
    □ Allows the user to ‘understand’ the data-flows in the system
  □ **Scripting** engine
    □ Allows for creation/execution of test-scenario’s
  □ **Capture** module
    □ Allows for the creation and visualization of a data-capturing time-line
  □ **Regression-test** execution module
    □ Allows for repeatable batch-execution of test-scripts
    □ Allows for interactive as well as headless execution modes
2.2 Architecture Overview

Scripting Engine
DDS-DSL #!jys #!jython
Exec Edit

Capture-Module
Timeline

System Browser

DDS Connection Module
3. Requirements
3.1 Non-Intrusiveness (1)

☐ **Requirement**: Black-box approach:
   - The DDS-system and its applications should be left ‘untouched’ w.r.t.
     - Configuration: system shall NOT require any (pre-)configuration
     - Execution: execution order/location shall not be influenced
     - Performance: test tooling shall not degrade system performance
   - The test tool shall be dynamic e.g. not requiring and pre-configuration (of type-descriptions, etc.)

☐ **Analysis**: Generic DDS context
   - DDS components are **decoupled** (in space/time) ‘by nature’
   - DDS dynamically **discovers** all (logical) DDS entities
   - DDS has standardized ‘built-in topics’ that capture that discovered ‘**meta-data**’
   - Yet DDS currently lacks (standardized) **dynamic API’s** and runtime topic/type creation

☐ **Analysis**: OpenSplice DDS context
   - OpenSplice DDS has **extended meta-data** that includes type info
     - Allowing for dynamic creation of readers/writers/topics (XML-based dynamic API similar to X-types)
   - OpenSplice DDS built-in-topics include **mapping-info** between logical and physical entities
     - Additional **CMParticipant** built-in topic that relates participants to processes running on particular nodes
3.2 Non Intrusiveness (2)

- **Requirement**: Target Platform Execution & Connection
  - The tool shall **NOT** be required to run on the ‘target-system’
    - From platform-support perspective (*platform might not support Java/HMI*)
    - From resource-usage perspective (*tool induced CPU and/or Memory load*)
  - The tool shall therefore be able to run on a **remote** computer
    - Yet without implying that all data is routed to that computer

- **Analysis**: Generic DDS context
  - Need to **decouple** the DDS_interactions from the scripting/analysis/visualization ‘engine’
  - Currently **no standardized solutions** for ‘remote DDS-access’ by a tool

- **Analysis**: OpenSplice DDS context
  - OpenSplice already has a **SOAP-based agent** (used by its ‘tuner’)
    - That exposes the dynamic “Control & Monitoring” (**C&M** API) to a remote application over SOAP
    - Which includes the ‘normal’ yet **dynamic** DCPS-API’s for reading/writing that are sufficient for the test-tool
    - As well as **specialized C&M functions** to ‘inspect’ DDS-entities and even change their QoS settings
      (as used by the OpenSplice Tuner whitebox testing tool)
Example: connection modes
Black- & White-box
3.3 Overall Test Nature

☐ **Requirement**: Regression Testing
  ☐ **Repeated** testing
    ☐ The tool shall have the ability to create test-scripts
    ☐ The tool shall have the ability to execute batches of scripts
    ☐ The tool shall have the capability to capture/maintain regression results

☐ **Requirement**: Integrated Test environment
  ☐ The tool shall **Integrate** browsing, capturing/manipulating, scripting
    ☐ The tool shall have the ability to create test-scripts from interactions
    ☐ The tool shall allow for one-clock monitoring
    ☐ The tool shall integrate all components to an intuitive look&feel
Example: Regression (batch-)testing
3.4 Scripting

☐ Requirements: Script ‘engine’
  ☐ The tool shall contain an intuitive and powerful scripting-engine
    ☐ To capture, monitor and inject DDS-data
    ☐ To define ‘business-logic’ to process the data

☐ Analysis: DDS ‘DSL’
  ☐ There’s a need for DDS-Domain Specific Language (DSL)
  ☐ Yet that can be ‘enhanced’ by available scripting languages such as Javascript, Python, Ruby
  ☐ A java-based tool-suite greatly facilitates inclusion of free interpreters (Jython, JRuby)

☐ Requirement: Script generation
  ☐ The tool shall facilitate easy script generation / editing

☐ Analysis
  ☐ The tool shall facilitate creating scripts out of observations
  ☐ Allowing to reproduce ‘live’ interactions for (regression-)test purposes
Scripting: JavaScript

```javascript
let c = ct;
collision = true;
colx = Math.round((x1+x2)/2)
coly = Math.round((y1+y2)/2)

return colx;

function get_yv(){
return coly;
}

function get_col_size(){
return Math.round(colsize);
}

get_yv();

repeat Circle 0.05 0 {
    color -> BLUE,
x -> 'get_x1()<c>','
y -> 'get_y1()','
shape_size -> 30,
}
```
Example: auto-scripting (1)
Example: auto-scripting (2)
3.5 Browsing

☐ **Requirement:** system browsing
  ☐ The tool shall allow **browsing** of the system-under test
    ☐ From a logical perspective: “what participants publish/subscribe what”
    ☐ From a physical perspective: “what applications are running where”
  ☐ The tool shall allow detection of **QoS conflicts** that prevent communication

☐ **Analysis:** DDS context (using standard built-in topics)
  ☐ Built-in topics provide all information for Building **logical views**
    ☐ Topic ➔ participants ➔ publishers/writers or subscriber/readers
    ☐ Participant ➔ publishers/writers or subscriber/readers ➔ Topics
  ☐ Built-in topics provide all information to detection of **QoS conflicts**

☐ **Analysis:** OpenSplice DDS context
  ☐ The dedicated OpenSplice **CMParticipant** built-in topic allows to create a **physical view**
  ☐ Discovered advertised port-numbers of our SOAP-agent allow to **spawn white-box ‘tuners’**
    ☐ That ‘connect’ to individual and/or groups of federated applications
Example: Testing ishapes: browser
Example: QoS conflict detection
3.6 Capture & Presentation

☐ **Requirement:** Capturing interactions
  ☐ The tool shall support **one-click** monitoring
  ☐ the tool shall create a **timeline** of all monitored data
  ☐ The tool shall provide manual or scripted **charting** of monitored data

☐ **Analysis:** generic DDS context
  ☐ Built-in topics provide the basic information for one-click monitoring
    ☐ The topics and their default topic/QoS policies
    ☐ The readers/writers and the QoS they’ve applied (including **Partition QoS**)
    ☐ The tool can then either use the topic-defaults or query for specific QoS settings

☐ **Analysis:** OpenSplice DDS context
  ☐ OpenSplice C&M API allows **dynamic creation** of readers/writers

☐ **Analysis:** free/open-source visualization options
  ☐ The Java environment offers libraries such as **Jchart2D** for visualization
Example: one-click monitoring (1)
Example: one-click monitoring (2)
3.7 Analysis & Manipulation

☐ **Requirements:** analyze **Timeline** of obtained data
- The tool shall allow to analyze & manipulate individual samples
  - Display Sample Data and sample Info (meta-data)
  - Read / Modify / Write samples
- The tool shall allow to analyze **streams** of topic samples
  - Browse the timeline of samples
- The tool shall allow to **Chart** captured (read/written) data
  - Interactive as well as from script
  - Including ‘scatter plots’

☐ **Analysis:**
- Configurable **time-line**
  - Selectable fields of monitored topic exchange (inclusive time & source of publication)
  - **time-based**, **topic-based**, **key-based** browsing through timeline
- **Virtual attributes** can greatly enhance interpretation of timeline
  - Use script-engine power to define virtual attributes (out of existing attributes)
- **Real-time** charting
  - Using existing open-source charting-libraries such as Jchart2D
Example: virtual fields

```python
nextfield
minY
Circle
Sample.getFieldValue("y") * -1
nextfield
minY
Square
Sample.getFieldValue("y") * -1
nextfield
minY
Triangle
Sample.getFieldValue("y") * -1
```
4. Summary

☐ Tool developed ‘by testers for testers’

☐ As a complementary tool to available white-box tuning

☐ Based on the open-source C&M API of OpenSplice DDS

☐ Exploiting the Java-platform and open-source plugins
  ☐ Javascript, jPython, jRuby script engines
  ☐ Jchart2D, Log4J, Swingx and RSyntaxTextArea libraries

☐ Enhanced with a ‘browser’ and productized by PrismTech
  ☐ Available as “OpenSplice Tester” with V6.1
4. “Tester” Live Demo
DDSTouchStone: DDS benchmarking suite

- One generic DDS application called ‘touchstone’
  - Configured with two parameters only:
    - Application-Id for identification of the touchstone
    - Group-Id for identification of the group the touchstone belongs to
  - Assures easy deployment
    - Interfacing done by means of DDS topics
    - Input settings (“..DefTopics” in the TouchstoneCommands partition)
    - Output results (“..ReportTopics” in the TouchstoneReports partition)
    - Assures location, platform and language independence of deployment

- A Container for the four key players:
The four key players

- Transceiver
- Transponder
- Transmitter
- Receiver
Inter-Nodal Roundtrip Latency

☑ 2006 ‘reference platform’
  ☐ Dell blade-server (Opteron)
  ☐ Dell-powerconnect 5324 Gbit LAN

☐ Metrics
  ☐ X-axis = Kbytes
  ☐ Y-axis = uSec

☐ Regression-test graph by Tester