Tutorial: Load Testing with CLIF

Bruno Dillenseger, Orange Labs

Learning the basic concepts and manipulation of the CLIF load testing platform.
Focus on the Eclipse-based GUI.

Menu

Introduction about Load Testing and CLIF
Lab 1: CLIF Eclipse console set-up
Lab 2: Test deployment and execution
Lab 3: creating CLIF test plans
Lab 4: defining scenarios with ISAC
Lab 5: a web load testing project
Lab 6: using external data sets
Demonstration/Preview of advanced features

Bon appétit !

Introduction

Load Testing and CLIF

Why Load Testing?

The goal of load testing is to verify that a (computing) system can sustain a given flow of incoming requests, while still meeting given requirements:

- no crash, nor stability or consistence troubles
- performance-related Quality of Service/Experience (latency, response time, request throughput...)
- keep responding to all requests
- resilience to traffic peaks or deny-of-service attacks

What is Load Testing?

Several approaches to performance assessment:

- program static or dynamic analysis
- system modeling + analysis or simulation
- load testing: traffic injection and system observation

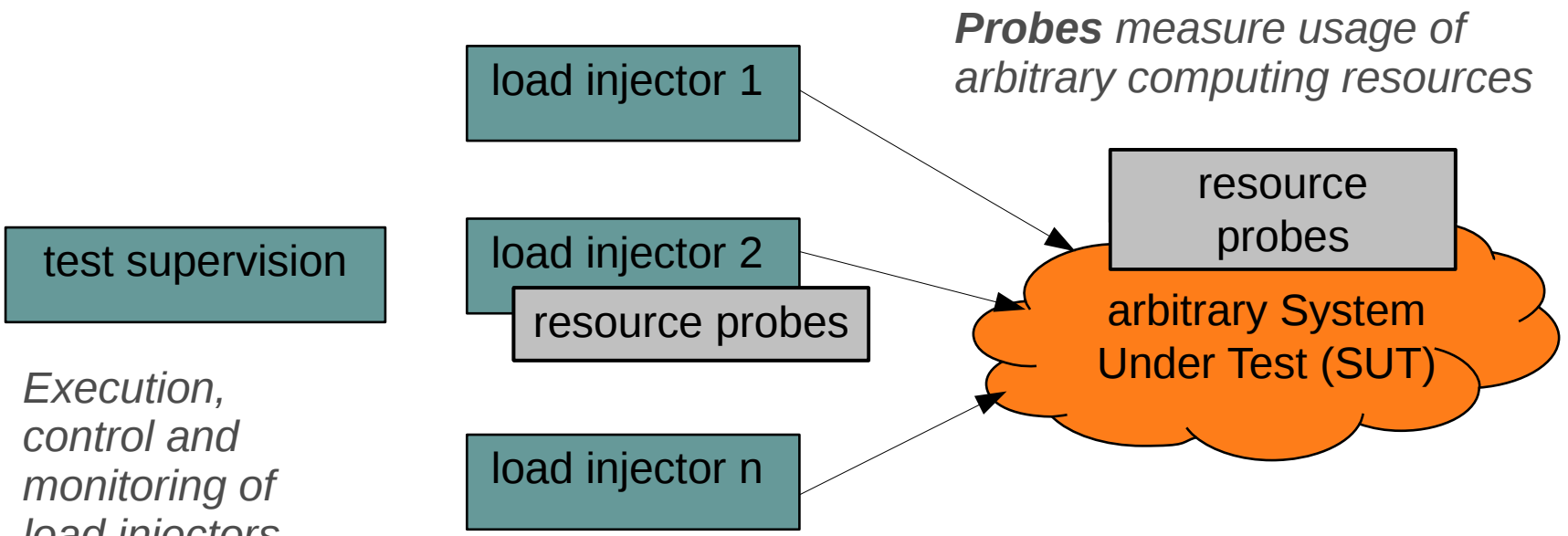
➔ pros:

- closer to the real conditions, although not strictly identical
- little knowledge about the system (black box)

➔ cons:

- time consuming and heavy testing infrastructure

Big Picture of Load Testing



***Probes** measure usage of arbitrary computing resources*

Execution, control and monitoring of load injectors and probes.

***Load injectors** :*

- *send requests, wait for replies, measure response times*
- *according to a given **scenario** defining the workload*
- *for example, emulating the load of a number of real users*

→ ***virtual users***

Common Traps

When you get your first measures, their interpretation is not as straightforward as expected:

- poor performance may result from bad configuration or bugs, in the SST or... in the load injection system
- the more information you have, the finer the analysis and understanding (probes, request-by-request details and profiling)
- but, the more you can get lost, handling a huge amount of measures

CLIF is a Load Injection Framework

CLIF is a Java software dedicated to load testing

- adaptable et extensible
  - open source (ObjectWeb project created in 2002)
  - any system under test (protocols, probes...)
  - any workload definition and execution support
  - any user interface (GUI, Eclipse, batch, embedded)
  - component-based architecture for minimal adaptation effort (OW2's Fractal model)
- high power
  - distributed load injection
  - no limit in terms of virtual users number

Lutèce d'Or 2007 award
best open source project lead by a big company

CLIF's Basics

A **TEST PLAN** specifies a list of probes and injectors to be deployed for a test

- **PROBES** monitor usage of arbitrary resources
  - provided by CLIF for Linux, MacOSX and Windows: cpu, memory, network, jvm, disk
  - arbitrary custom probes based on JMX, SNMP or any other middleware...
- **INJECTORS** send requests on the SUT
  - ISAC environment provided by CLIF: IsacRunner scenario execution engine
  - arbitrary custom load injector

Lab 1

CLIF Eclipse console set-up

step#1: CLIF Eclipse console set-up

- install JDK 6 (Sun/Oracle's JDK or OpenJDK)
- 2 options for CLIF installation:
  - CLIF plug-ins in Eclipse 3.5 (Galileo) PDE
    - full-featured console, recommended
    - install Eclipse 3.5 PDE for your operating system
    - unzip CLIF plug-ins in *path_to_your_eclipse_root*/dropins
  - CLIF standalone Eclipse RCP console
    - simpler console with minimal features
    - unzip CLIF RCP console for your operating system

step#2: CLIF perspective and views

- Choose Clif perspective
  - either Window>Perspectives>Clif Perspective
  - or Window>Perspectives>Other... Clif Perspective
- The CLIF perspective is a set of 4 views:
  - Navigator, Monitor, ClifTreeView, Console
- To restore a view
  - Window>Views

step#3: main CLIF wizards

CLIF Project

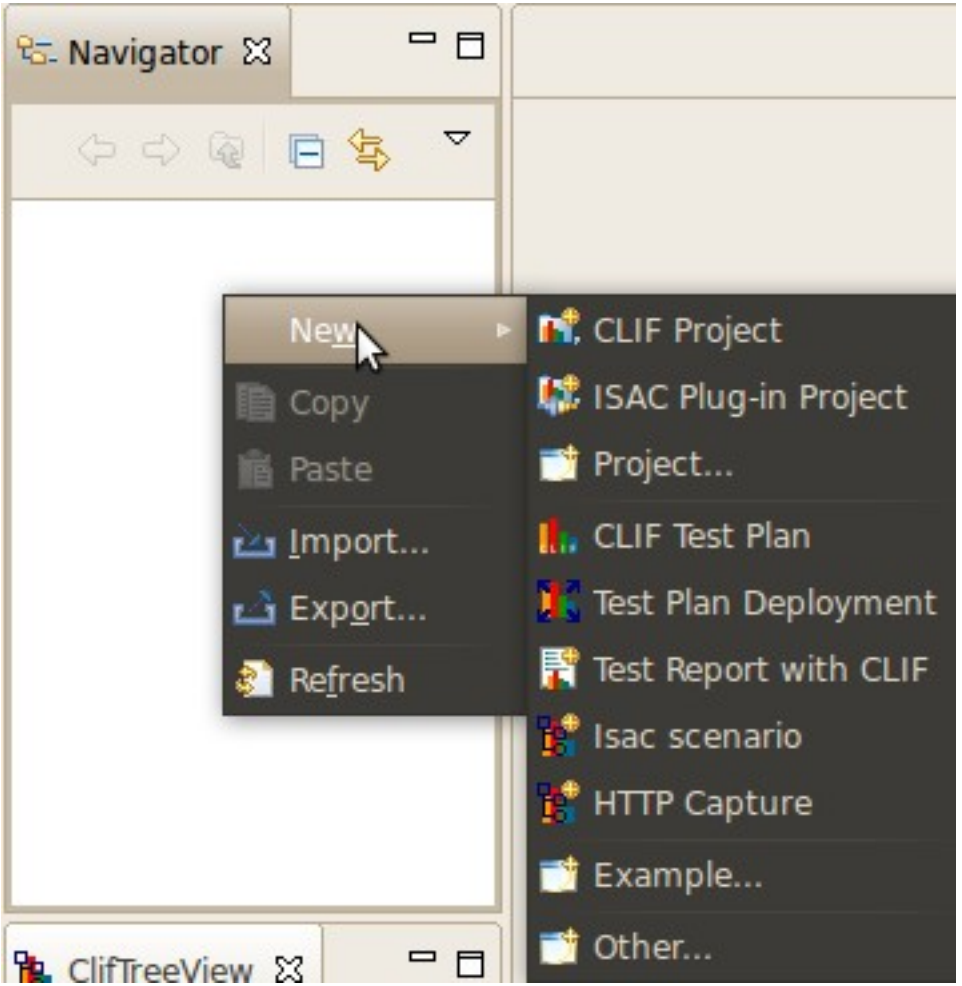
- create a new directory (aka container) to define test plans and scenarios

CLIF Test Plan

- creates a test plan file

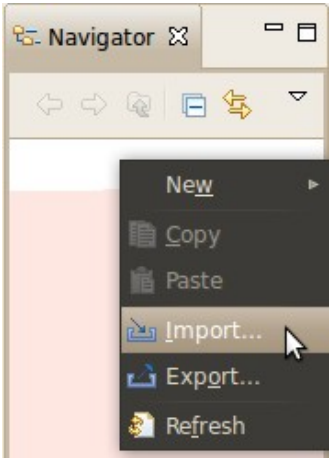
Test Plan Deployment

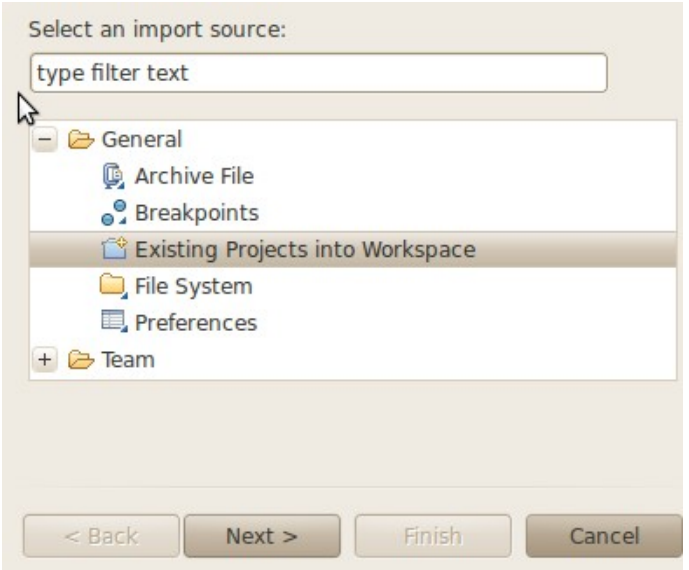
- deploys a test plan

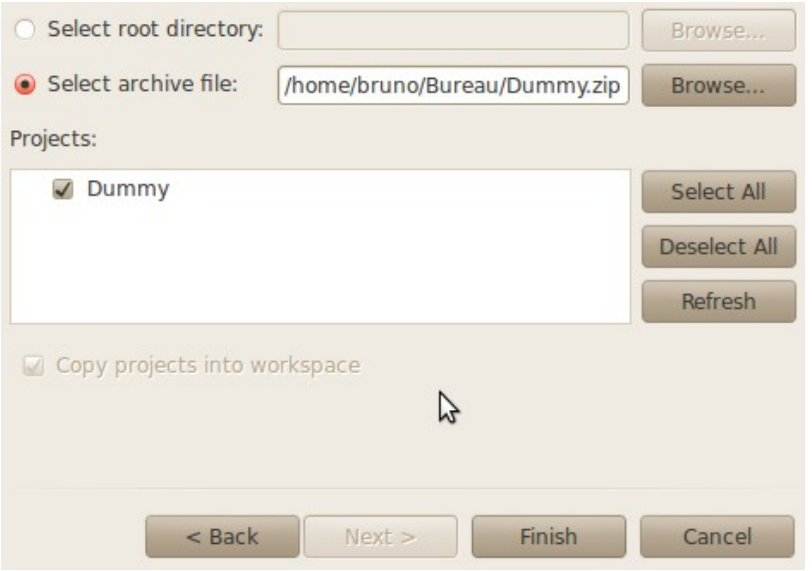


step#4: import 'Dummy' project

- download Dummy.zip file (do not unzip!)
- right-click in Navigator view>Import...
- General>Existing Projects into Workspace [Next >]
- Select archive file>Browse... *Dummy.zip* [Finish]





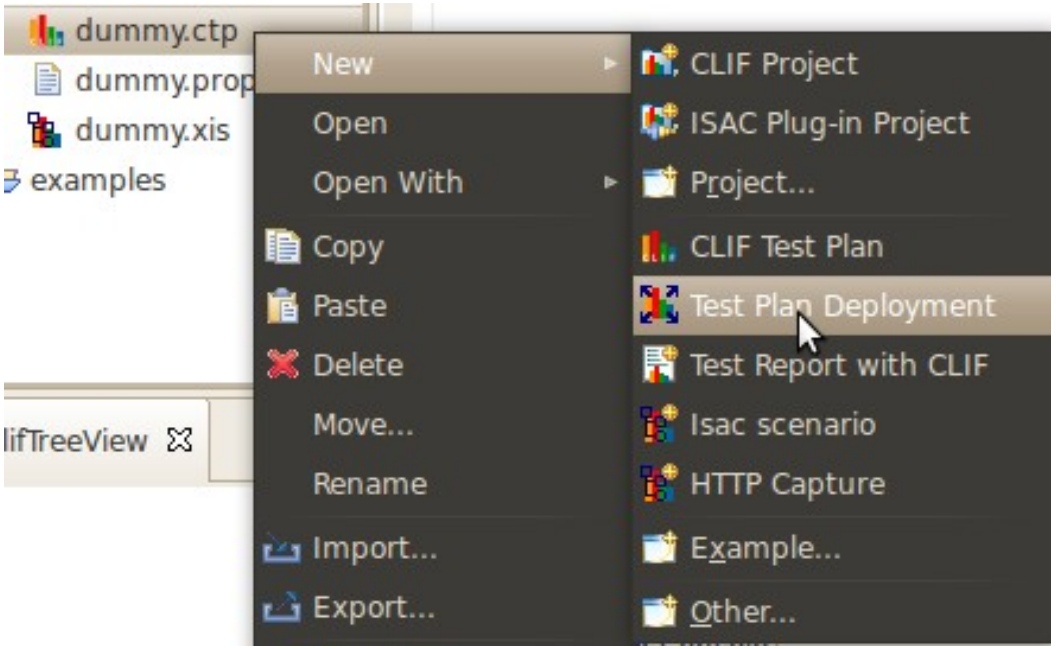


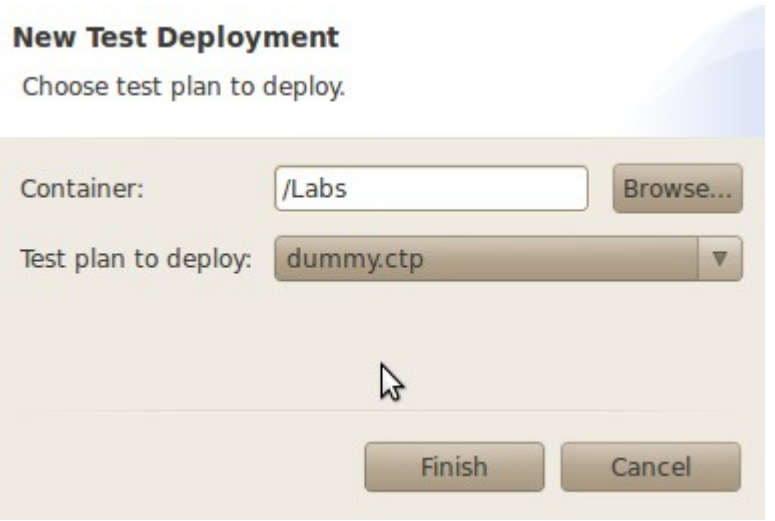
Lab 2

Test deployment and execution
Measures collection and browsing

step#1: deploy dummy.ctp

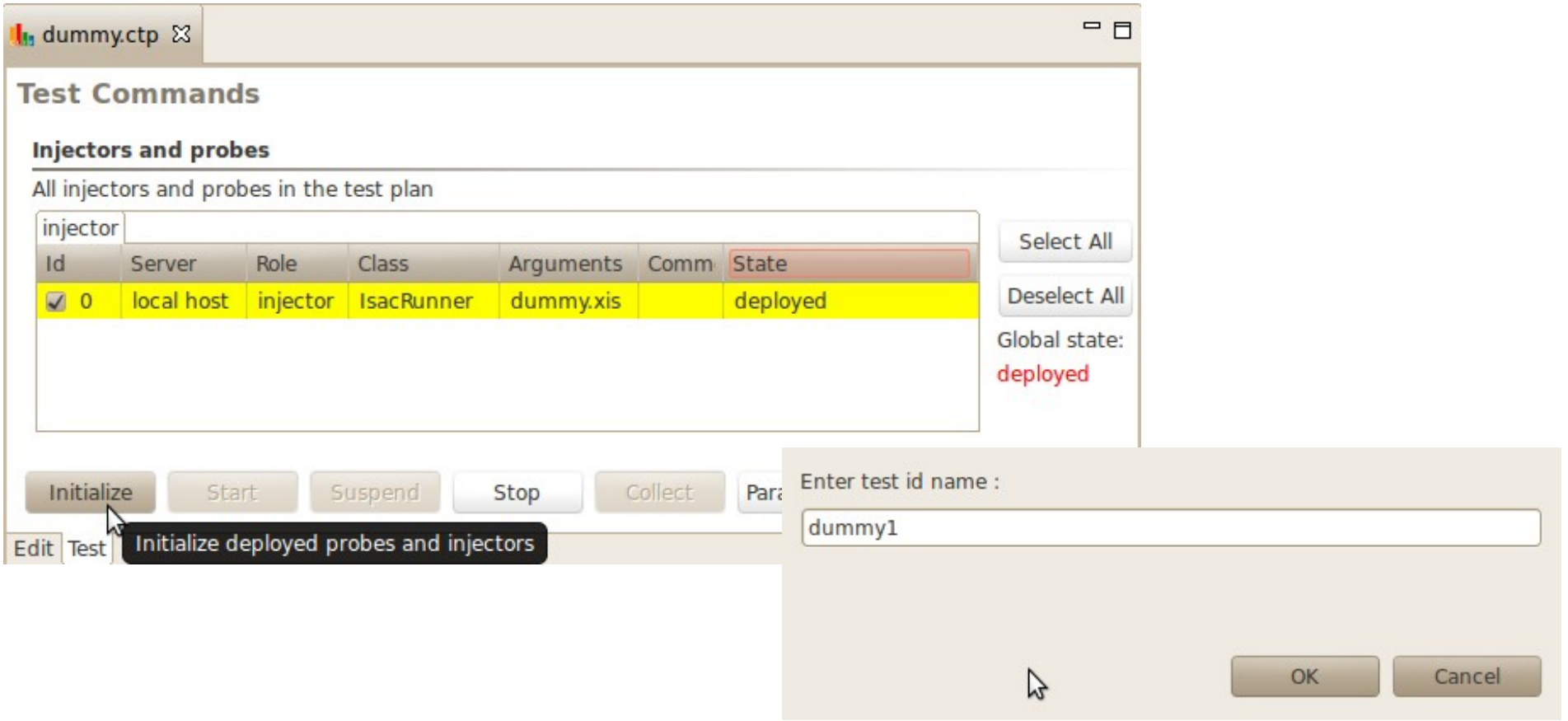
- In Navigator view, right-click on dummy.ctp in Dummy project
- New>Test Plan Deployment>... [Finish]





step#2: test initialization

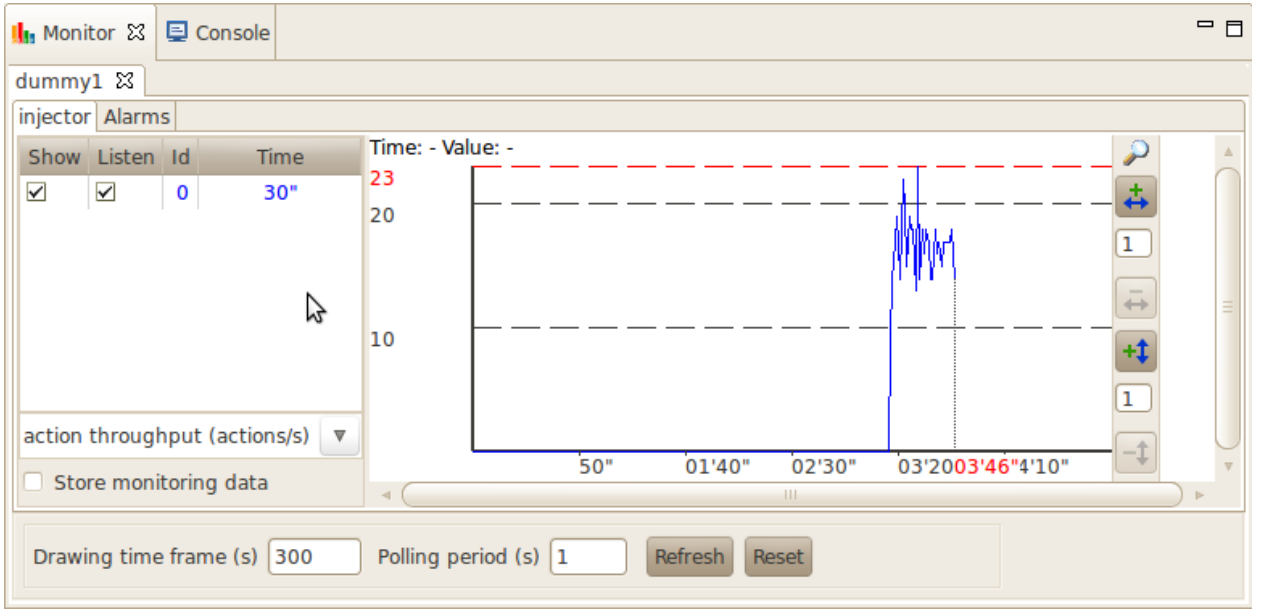
Click on [Initialize] and choose a test run name



step#3: execution and supervision

Once initialization is done, the Monitor view activates and opens a new tab for this test run

- Select the action throughput metric in the 'injector' tab
- Click [Start]
- try [Suspend] and [Resume]
- wait for completion or [Stop]



step#4: collect measures

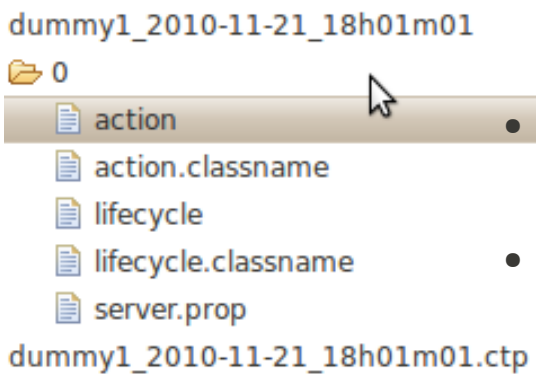
Click on [Collect] to gather all measures in the 'report' directory in the 'Dummy' project.

- Note 1: the collect operation in instantly done here, because the load injector in test plan 'dummy.ctp' is deployed in the console itself.
  (see “Server”: “local host”)
- Note 2: this test can be re-executed repeatedly by clicking Initialize, Start and Collect again and again

step#5: browse the measures

Refresh the 'report' directory in 'Dummy' project (right-click>Refresh) and browse its structure:

- test run name_execution-date_execution-time
  - injector or probe identifier in the test plan ('0' here)



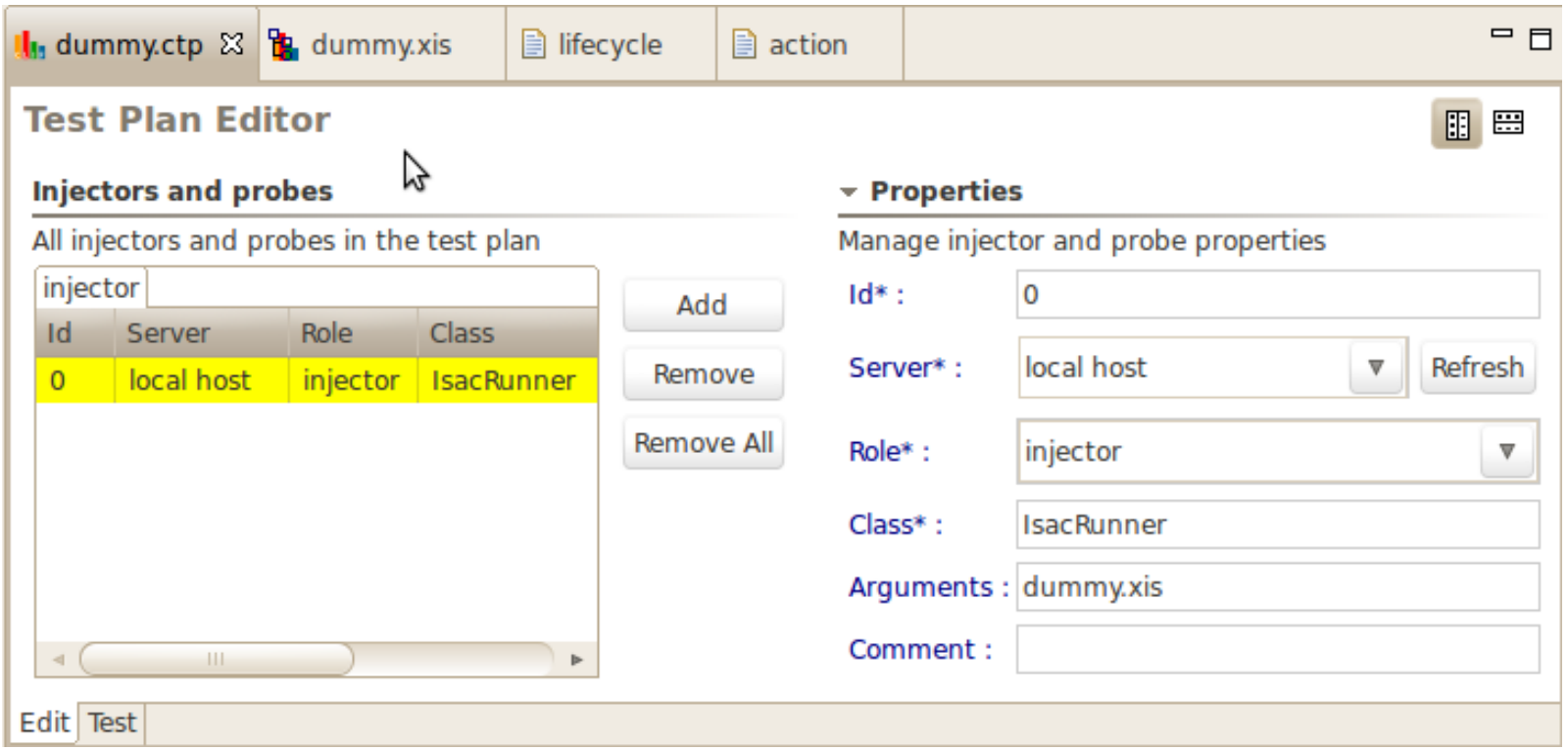
    - action: text file of values separated by commas; each line is a request report recording response times and request success
    - lifecycle: a trace of the test status changes: initialized, started, suspended, resumed, aborted, completed, stopped.
    - server.prop: a reminder of all the Java Virtual Machine system properties
    - .classname files (ignore)
- test run name_execution-date_execution-time.ctp

Lab 3

CLIF test plans
Probes
Distribution support with CLIF servers

step#1: test plan edition

Click on the Edit tab at the bottom of the 'dummy.ctp' test plan editor.



Focus on Test Plan Contents

A test plan is a list of load injectors and probes

- **Id**: a unique name identifying this injector or probe in this test plan (defaults to a generated integer)
- **Server**: the place (CLIF server name) where to deploy this injector or probe. “local host” is a default CLIF server embedded in the console.
- **Role**: load injector or probe
- **Class**: 'IsacRunner' is the execution engine for workloads defined with ISAC scenarios (consistent with injector role)
- **Arguments**: the ISAC scenario file (consistent with 'IsacRunner' class)
- **Comment**: free comment for this injector

step#2: define probes

In the 'dummy.ctp' editor, click on [Add] and define 3 probes:

- respectively with classes cpu, jvm, memory
- with explicit unique identifiers (e.g. CPU, JVM, RAM)
- deployed on server “local host”
- with arguments: 1000 30

Save, then deploy, initialize and start. Monitor the probes. Finally, collect and browse the measures.

Focus on Distributed Test Plans

The **CLIF console** is the central point of control:

- deployment of test plans
- supervision of probes and injectors (test execution)

CLIF servers host probes and injectors:

- default “local host” server embedded in the console
- arbitrary name on local or remote computer, registered in the **CLIF registry**
- dependencies are loaded from the **CLIF code server** embedded in the console

step#3: CLIF server set-up

- install apache ant 1.8.1 utility
  - unzip and add *path_to_ant*/bin to your PATH
- install CLIF server
  - unzip CLIF server binary distribution
  - open a command interpreter and set current directory to CLIF server's root
  - run command “ant config”
    - please enter the registry host: **localhost**
    - please enter the registry port number: **1234**
    - please enter the code server host: **localhost**
    - please enter the code server port number: **1357**

step#4: run a CLIF server

- In the CLIF console, check that the registry is running by reading the Console view content:

```
Creating a CLIF Registry... Fractal registry is ready.
registry@localhost:1234
```

- In the command interpreter, run command “ant -Dserver.name=myServer server” and wait for the following message:

```
[java] CLIF server myServer is ready.
```

step#5: write a distributed test plan

Set the target CLIF server of the jvm probe and injector to myServer:

- edit 'dummy.ctp'
- select the injector
- click on [Refresh] button on the “Server” selection line: 'myServer' appears - select it
- select the jvm probe, and change the Server (needless to refresh the list of servers)
- save, deploy, run (init, start), monitor and collect

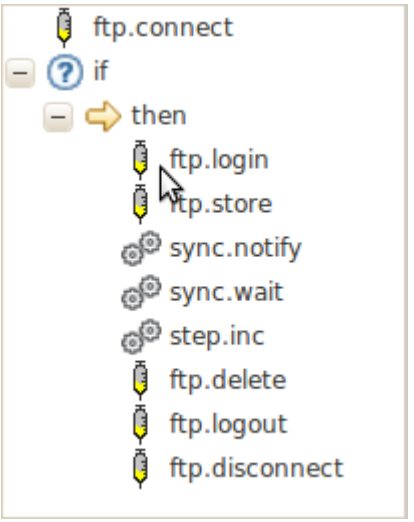
Lab 4

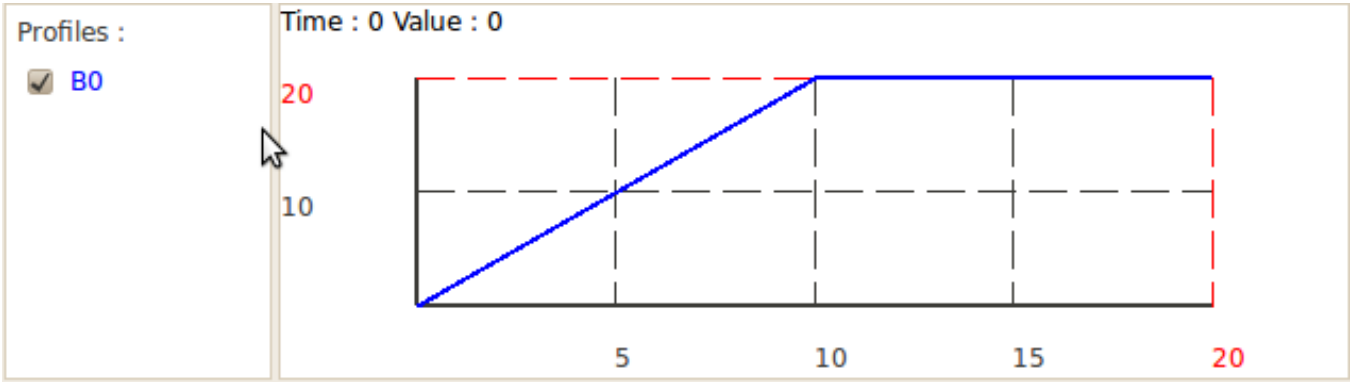
Defining workload with ISAC scenarios

Defining Workload with ISAC

ISAC is a Scenario Architecture for CLIF

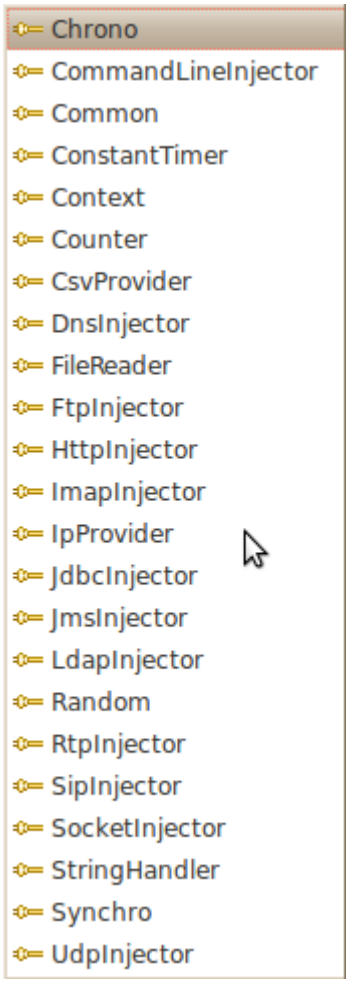
- formal definition of virtual users behaviors
  - requests on the SUT, think times (inactivity periods)
  - control logic constructs: if-then-else, while, probabilistic branching, preemption condition
- one load profile definition per behavior
  - # of active virtual users as a function of execution time





ISAC extensibility through plug-ins

ISAC is generic and extensible through "plug-ins"



Each behavior defines a generic logic, which imports and uses **plug-ins** providing:

- **think times** (constant, random with any kind of distribution, arbitrarily computed...);
- **samples** (requests) on the SUT, according to the appropriate protocol;
- **conditions** used by if-then-else, while and preemption statements;
- **control** operations to manage ISAC plug-ins;
- **data provisioning**
  - to play scenarios with external data sets
  - to exchange data between plug-ins

step#1: look at dummy.xis imports

'dummy.xis' is the scenario used by the injector defined in 'dummy.ctp' (see injector's argument)

- open the file from the Navigator view
- click on the 'Import' tab at the bottom part of the editor and look at the used plug-ins:
  - Random, Common, Context
  - click on Context to see its parameters, and have a look at file 'dummy.props' in the Dummy project
  - click on [Add] button to see te list of available ISAC plug-ins; click on [Cancel] button to close this list.

step#2: look at behavior B0

Behavior B0 simulates requests with random response times and random think times.

- Click on tab “Behavior B0” to the see the virtual users' activity definition. Click on each line. Note the use of ${*plug-inId*:*variable*} statements.
- Check the 'Load profiles' view and edit the load profile by clicking on the [Modify] button
  - **time** is in seconds, **population** is the number of active virtual users (vUsers) at this time
  - “**force stop**” is enabled: vUsers may be stopped without waiting for their full behavior completion

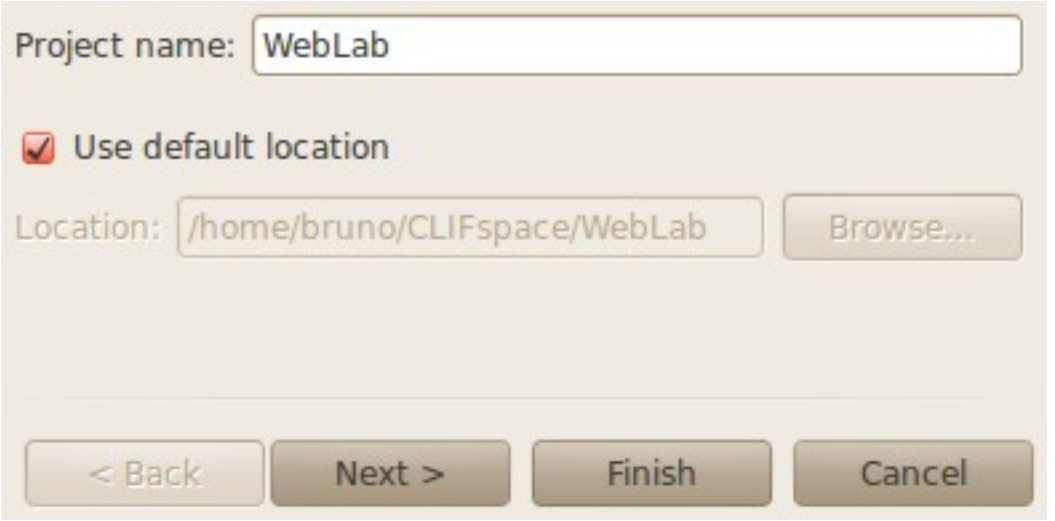
Lab 5

a web load testing project

step#1: create a CLIF project

Right-click in the Navigator view and run the “New>CLIF Project” wizard

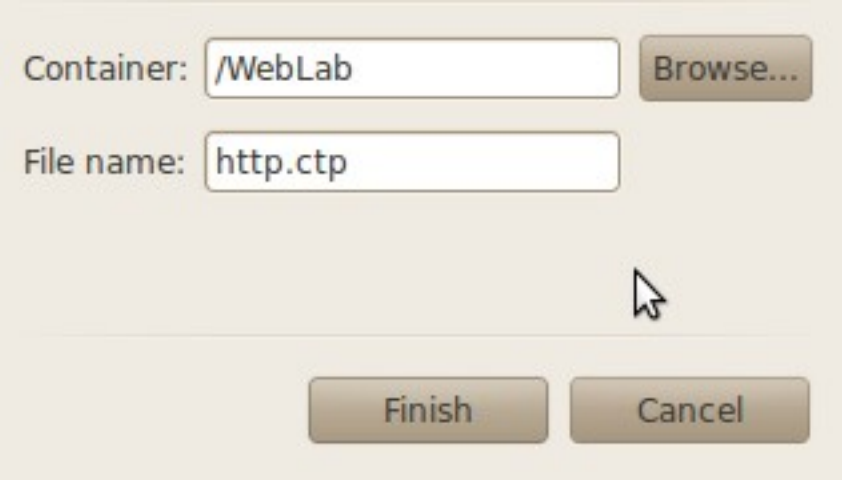
- enter a project name (e.g. WebLab)
- keep the default location
  - for your information, note the project path
- just click on [Finish]
  - [Next >] gives access to advanced settings



step#2: create a test plan

Right-click on 'WebLab' project and run the “New>CLIF Test Plan” wizard

- enter a test plan file name ending with .ctp (e.g. http.ctp) and click on button [Finish].
- The new test plan appears in a test plan editor.



step#3: add an IsacRunner injector to the test plan

In the 'http.ctp' test plan editor, click on button [Add] and enter the parameters:

- Id: keep default '0' or change to a more explicit name (e.g. webclients)
- Server: keep 'local host'
- Role: injector
- Class: IsacRunner
- Arguments: http.xis
- Comment: HTTP traffic generator

Save the file.

step#4: create a scenario

Right-click on project 'WebLab' and run “New>Isac scenario” wizard

- Enter a scenario file name ending with .xis, as set in the 'http.ctp' test plan ('http.xis') and click on button [Finish].
- The new scenario appears in an ISAC scenario editor.

step#5: edit virtual users' behavior

In scenario editor 'http.xis':

- import plug-ins HttpInjector, ConstantTimer, Counter
  - rename their import id with a shorter and more explicit id: 'http', 'timer1s', 'loop5'
  - initialize 'timer1s' to 1000 ms duration
  - initialize 'loop5' value to 5 (let 'shared' option disabled)
- edit behavior 'B0' to implement the following:
  - each vUser should make 5 HTTP GET requests to the sample web application, with a 1 req/s throughput;
  - start with no vUser, ramp-up to 10 vUsers during 5s, and then 10 vUsers plateau during 25s.

step#6: deployment and execution

- Deploy test plan 'http.ctp'
  Important notes:
  - on each change in http.xis, **the test plan must be redeployed**
  - deployment is possible only when there is **no active (deployed, initialized or running) test**.
- Initialize and Start the test
- Monitor response times, request throughput...
- Collect, refresh and browse directory 'report'

step#7: web capture and replay

Right-click on the 'WebLab' project and run wizard “New>HTTP Capture”

- enter a file name for the scenario, e.g. capture.xis
- click [Next>] and [Validate Configuration]
- set your web browser's proxy to localhost:8090
- in the capture wizard, click [Start Recording]
- navigate through the sample web application
- in the capture wizard, click [Stop Recording] and [Finish]
- in the 'WebLab' project, open file 'capture.xis', and set a load profile for behavior “session”
- define and run a capture.ctp test plan with capture.xis

Lab 6

Using external data sets in ISAC

step#1: external properties

Goal: define the root of requests URLs in a separate properties file, using the Context plug-in.

- right-click on the 'WebLab' project and run wizard “New>Other...>General>File”
- enter a file name for the properties file, e.g. 'http.props'
- edit the file and set two properties: host=... port=...
- edit file 'http.xis': import plug-in Context, give it a simpler import id (e.g. const), add a field in the “Load properties file” section, and enter 'http.props' in this field
- edit behavior B0, select the 'http.get' line, and change the URI parameter to: http://${const:host}:${const:port}/MyStore

step#2: external data set

Goal: define a list of parameters in a separate file, using the CSVProvider plug-in.

- create file 'http.items' and enter one value per line:
  - My%20favorite%20Elvis%20songs
  - Forgetting%20Sarah%20DVD
  - Christmas%20CDs%20collection
  - New%20York%20Fair%201939
  - Collector%20DVD%20collection
- edit file 'http.xis':
  - import plug-in CSVProvider, give it a simpler id (e.g. items), and set the file name (http.items) and field name (e.g. item)
  - edit behavior B0, select the 'http.get' line, and concatenate string '?buyItem=${items:item}' at the end of the URI
  - add a call to items.next() in the loop to iterate on the items

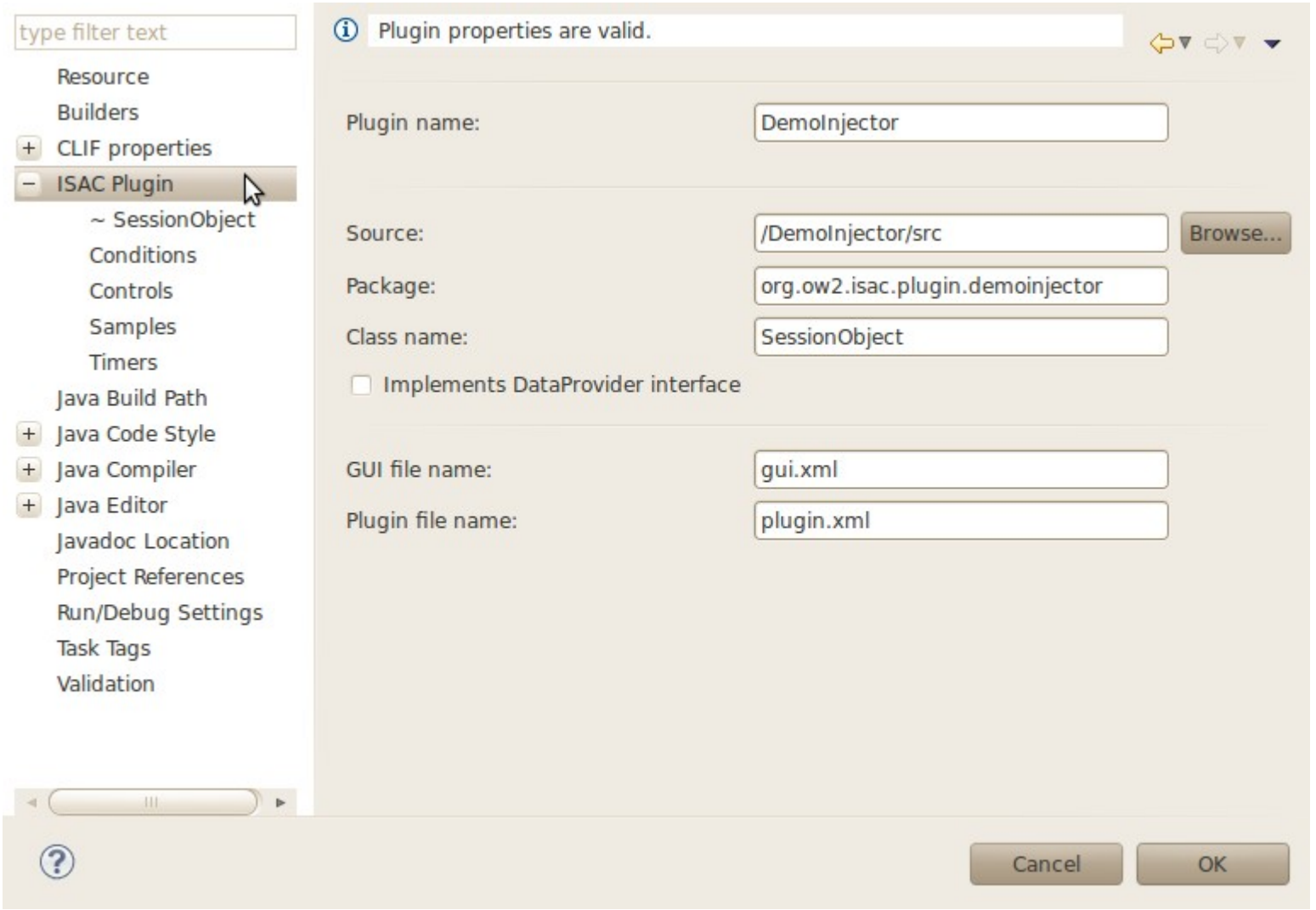
Demonstrations

CLIF project properties
Wizard for creating ISAC plug-ins
Alpha-version of CLIF reporting tool

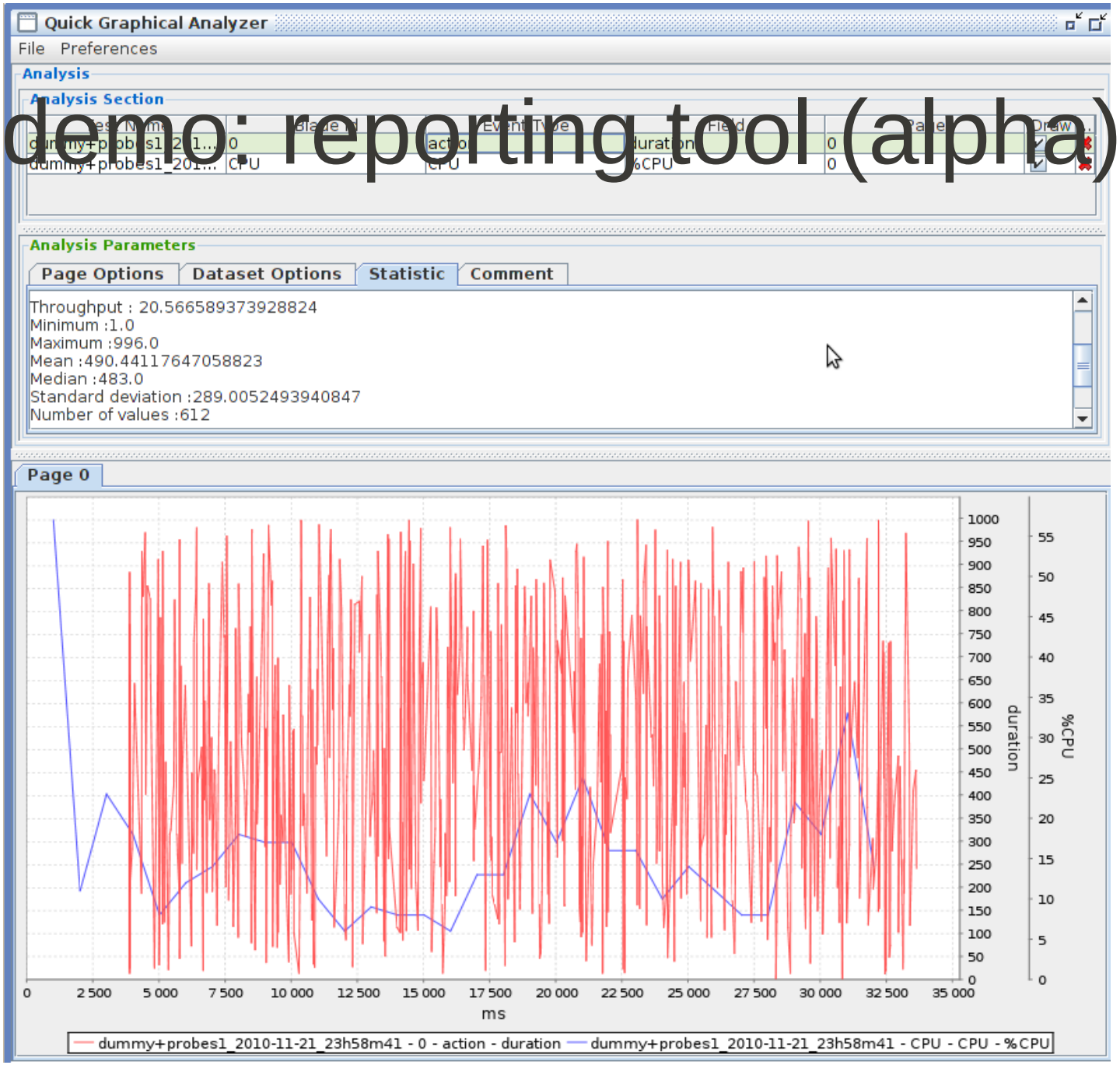
CLIF project properties

- Right-click on 'Dummy' project and select Properties option. Select 'CLIF properties'.
  - directory settings for measures and monitoring data storage
  - advanced settings for storage system tuning and network address selection in multiple subnetworks environments
  - custom system properties
  - CLIF registry and code server settings
  - ISAC execution engine tuning
  - JVM tuning (ignored by the console JVM)
- All these settings are **local**. You may copy file 'clif.props' to CLIF servers' etc/ directory.

demo: ISAC Plug-in Creation



demo: reporting tool (alpha)



That's all folks!

THANK YOU FOR YOUR ATTENTION

http://clif.ow2.org
clif@ow2.org