QL Services

- Aditya Dada

Architecture

There are two parts to the Testing Harness

a) A build framework that executes the tests and reports the results

b) The value added-features that work on top of the build framework to give ease of use

The Build Framework

Build Utility

The build utility used for our test harness is ANT (http//ant.apache.org/)

Reasons for using ANT

- Platform independent. Thus there does not need to be any extra coding for different platforms for majority of the operations.
- XML based. it is easy to read, write and understand. The learning curve for learning ANT, writing your own build scripts (build.xml files) is very low.
- Easier to debug than makefiles and shell scripts. since the learning curve for ANT is low, and the files written in xml are very easy to read and understand, it takes substantially lesser amount of time for an entire team to become familiar with ant. also, since ant is parsed, extra spaces do not break the build scripts, like makefiles break. also, there are a lot of editors that support xml parsing so you can see the xml errors easily.

Reporting

We use an in-house reporting utility which gets called from the client of the test and which generates report in the Compliant Output Format (COF) in xml at the root of the workspace.

History behind reporting mechanism being used

Our group used to use a reporting called statusU which was very easy to use and that output the report in text format at the console. A need was felt to have a better reporting mechanism using xml, since xml was the emerging standard in new technology at the time, so that we should be able to generate different formats of reports by just parsing the xml report in any way we desire. Thus, a reporter utility component was created by the team. This utility was how ever heavy weight and not as easy to use as the statusU. Our particular team did not want to let go of the ease of use and thus came up with a

reporter Adapter, a utility which used the xml from the reporter utility to report but could be instantiated and used just like statusU.

Harness Structure

```
appserv-tests
|--/config
     |--common.xml [common targets used by all e.g. deploy-common]
     |--properties.xml [common properties used by all e.g. ASADMIN]
|--sqe-tests
     |--/ejb
          |--/stateless
               |--build.xml [module level and common targets]
               |--build.properties [module level properties]
```

Common Services

Ability to add automation for tests that use:

- Application Clients
- Web Client
- Standalone Clients
- EJB/J2EE/Corba Technologies
- Shell Scripts

Ability to Execute:

- Entire Test Suite
- One component/module's tests
- Single Test
- One target for all tests (e.g. Clean everything, then setup everything)\
- Custom Targets
- Using a single command
- Dependencies before running targets
- Selectively running targets (if conditions are met)
- a shell script, be driven by a shell script.
- Tests for different products (PE/EE – DAS, Remote Server Instance)

Ability to Report:

- Passes, Fails and Did Not Run's
- in XML Format
- in HTML Format
- in Text Format

Debugging Capabilities using:

- Client (console) Logs
- Server Logs

Common and Local Properties and Targets:

- common.xml file
- common.properties file
- build.xml file
- build.properties file

Start and Stop Servers and Databases

- Ability to execute appserver CLI, java commands

Improvements needed for the Test harness:

- Incompatibility with Windows 2000 Professional (some SPs only).
- Memory leaks with ANT affects test runs
- No ability to re-run failed tests
- 'did not run' reporting needs to be more robust.
- No ability to capture server log/console log exceptions/errors for every failure/did not run
- No 'for-loop' capabilities available with the default ANT.
- No in-built ability to automate GUI applications.