Tests Configurations Management in Automation Testing

The writer is an independent consultant for automation testing in the fields of networking and J2EE applications. Guy_Arieli@hotmail.com. Tel: +972-54-7899446.

Abstract

The dynamics in the QA organization is not doing well to the automation project. The changing pressure on this organization doesn't enable the luxury of resources dedicated to automation. The bottom line is most of the automation projects are not going through the needed requirements analysis and design phases. It increase tendency to jump into the writing of the tests and having "quick results".
So it end up in situation were every test has its own configuration. Thousand tests with thousand configurations.
In this article I will present a mechanism that enables common configurations between tests.

Problems of usual situation

So what is the problem with the current status?

- It creates restrictions for tests execution order. Tests can't be executed independently and need other test to be executed before they do.
- When one test fails it can affect the other tests. It doesn't enable a reasonable recovery mechanism. So usually when one of the tests in the tests chain fails other tests will follow.
- You have more code and more code repetitions.
- You get longer tests execution time.

The severity of the problems grows when the project main target are system functional tests (and not unit or integration tests).

Configuration models

The 2 common options for configurations managements are:
Creating a bank of configurations and hierarchy of configurations.

Bank of configurations

In figure 1 the big bullets represents configurations. There is the root configuration. In this case there are 3 configuration linked to the root configuration C1, C2 and C3. The small bullets are tests. T1, T2 and T3 are bind to C1 configuration T4 is bind to C2 configuration and so on.

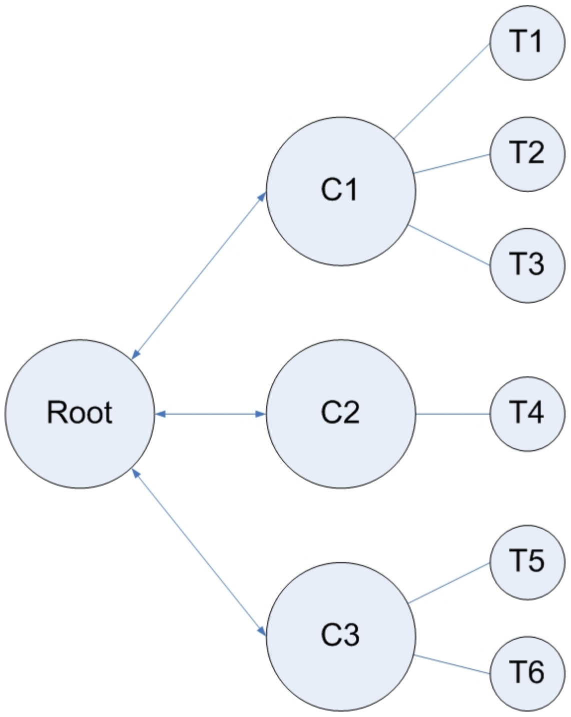

Figure 1

The root configuration is usually a minimal configuration that is easy to obtain.
Every configuration (C1, C2 and C3) define 2 processes. The first is moving from the root to the configuration and the second is moving from the configuration to the root.
I will call them setup and teardown (JUnit conventions).
Every test is bind to a configuration. The bind mechanism could vary.
The framework should be aware of the system current configuration status. Let's say we are in the root and we would like to execute T1 T2 and T5. First of all C1 setup will be executed, then T1 follow by T2. Then C1 teardown follows by C3 setup and T5.
It's important to understand that the movement between configurations should be done automatically by the automation framework.
Test as a guideline should not change the product/system configuration.
You can consider add additional process to the configuration that will be executed on test fail. The teardown-fail will be run every time a test under the specific configuration fails.
It will include severe actions (that usually consume more time).

For example if I would like to run T1 and T2. If T1 failed, C1 teardown-fail will be executed then C1 setup and then T2.

Hierarchy of configurations

In the hierarchy configurations model you create a tree of configurations (figure 2). As in Bank of Configurations, every configuration as the setup and teardown processes.
For example (look at the second diagram) the system is in the root and I would like to run T1 and T2. I will execute C1 setup run T1 then C2 setup and finally run T2.

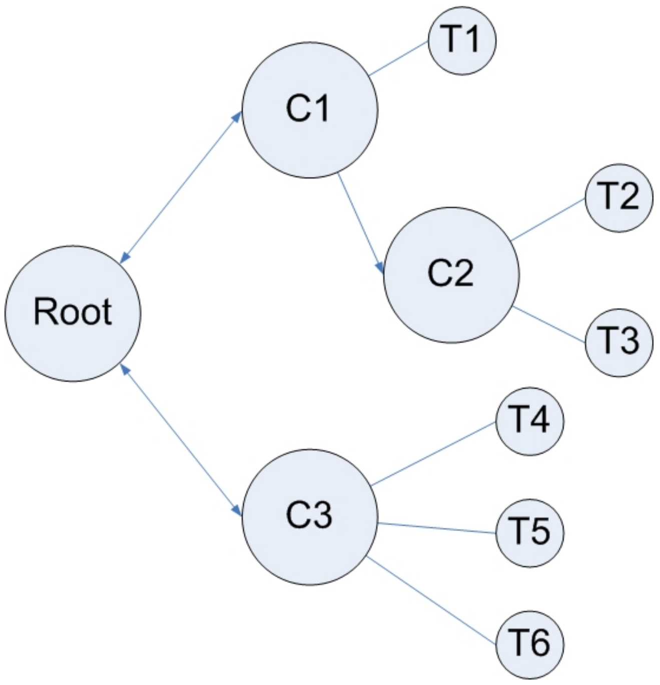

Figure 2

I recommend using hierarchy of configurations in 2 cases:

- When the configuration time is very long and time is a critical factor. By using a configuration tree the movement time between configurations will be shorter.
- When the number of configurations is very big and configuration tree will help you manage them.

Methods Benefits

I used the 2 models with great success in a few projects. I found the following benefits:

- It reduces the tests code in the project dramatically.
- It simplifies the writing of unit testing when the tests should run in the system environment.
- When one test fails it doesn't likely to cause other tests to fail.
- It shortens the over all tests set (suite) execution time.
- It's an excellent tool to support manual testing. In complicated systems the efforts invested in setting the condition for the tests usually greater then the testing itself. Such a tool can save a loot of time and efforts.

Implementation suggestion

Personally I'm in favor of JUnit. JUnit is excellent framework for unit testing and with few simple extensions it becomes excellent for integration and system testing as well. Tests configurations management is one of those extensions.
You should be aware that hierarchy of configurations contains the bank configuration model. So if you work on many projects it will be wise to implement the hierarchy of configuration and use it to the bank of configuration as well.
I define new interface with setup and teardown methods. Every instance of this interface will represent a configuration. I use the package mechanism of java as the base of the tree.
If I would like to implement the Bank model I just define one of the hierarchies in the package to be configuration hierarchy.
In order to make it a tool you should add a graphical view that will present the configuration tree. It should show the current state and enable the movement between configurations.