1. Presentation

This document proposes a possible implementation of a test campaign manager for dtest.
The suggested solution provides in one hand a mechanism to launch multiple tests sequentially, and on the other hand, a simple organization to manage most needs of test processes.
This project is almost independent of dtest and could perfectly be left aside from it. However, such an evolution in the dtest project could make it handier.

The three following themes are discussed in this document:

- The project's global concepts,
- Test campaigns,
- Tests implementation.

2. Concepts and terminology

A test *campaign file* is a XML file referencing different *tests* to execute. Some parameters can be included in the *campaign file*, which will be passed to the *tests* before execution. The proposed mechanism is generic and will match most testing needs.

A *test* is a python class matching a specific template (a few methods must be implemented, or inherited from the provided ParentTestClass).
As the repetitive testing needs for a given project implies repetitive code blocks, inheritance mechanisms should be used abusively during *tests* implementation.

The *tests manager* is a module that first parses the test *campaign file* to get the campaign's specific parameters and the list of the *tests* to launch.
Each *test* is then:

- Instanced,
- Given the campaign's file parameters,
- Initialized,
- Launched,

3. Test campaigns

The following describes the XML campaign's file structure.

The root element of the file is <campaign>, and contains the following elements:

- <parameter name="parameter_name" value="parameter_value"/>
    - A *campaign file* contains zero or more <parameter> elements.
    - The attributes name and value describe one parameter that will be passed to the *test* class instance before launching the test.
    - The parameters are added to a python dictionary, which suggests that no parameter element should have the same "name" attribute's value.
- <test_module path="/path/to/a/test directory/"/>

- At least one <test_module> element must be present in the *campaign file*.
- This element's path attribute contains a path to a directory containing a file named test.py, itself containing at least one *test* class.

Below is an example of test *campaign file*:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<campaign>

    <parameter name="server_ip" value="192.168.0.21"/>
    <parameter name="client_ip" value="192.168.0.42"/>

    <test_module path="/home/test/test1/">
    <test_module path="/home/test/test2/">
    <test_module path="/home/test/test3/">

</campaign>
```

4. Test classes

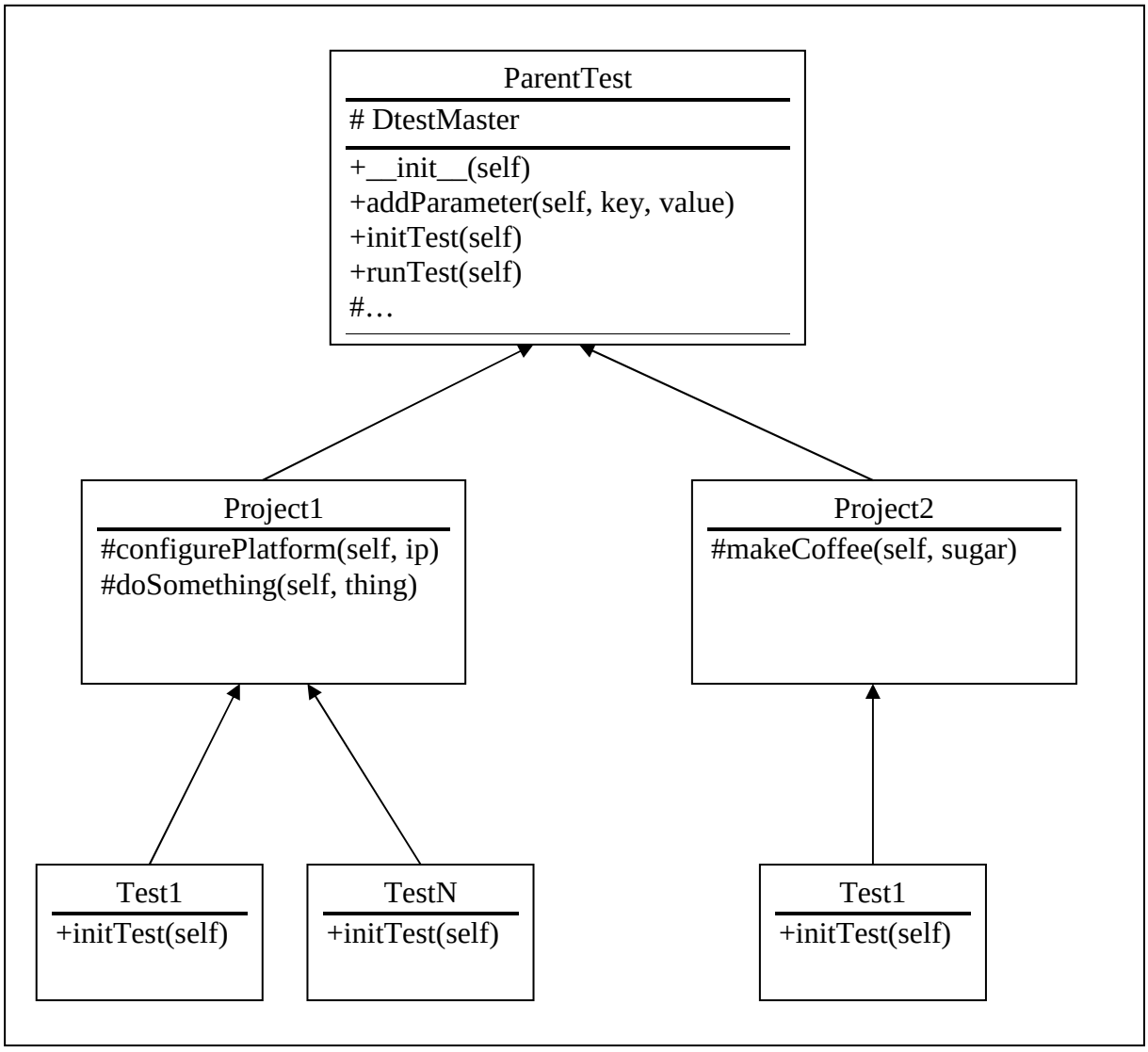
A test file's name is test.py. It is located in its own test directory (pointed to by the <test_module> element of the campaign file).

The file test.py must declare at least one class following the template below:

- A constructor with no parameters.
- An addParameter method with 2 parameters (key and value).
  - Invoking this method adds the key/value pair to an internal attribute which type is dictionary.
  - This method is used during the campaign file's parsing when a <parameter> element is encountered.
- An initTest method that does whatever you want to init your test.
- A runTest method with no parameters that launches the test.

The class could inherit from the provided ParentTestClass. This class could be used to implement functionalities commonly used during a testing process (execute a single remote command; transfer a file etc.).

The inheritance mechanism could be used the following manner:



In the example above, the ProjectN classes are only used to provide useful methods and will never be instanced. Here, the initTest method is used for:

- Dtest objects creation and setup
- DtestMaster setup

Usually the runTest method won't have to be redefined because its purpose is only to start the DtestMaster, and can consequently be defined in ParentTest.