What is DTest ?

lionel.duroyon@cert.fr
Stagiaire Onera DTIM

11 avril 2008

1 Introduction

Dtest is a distributed test framework written in the python language. It allows to execute and control distributed test sequences in order to do system testing. The goal of DTest is to verify that a distributed execution of different processes correctly matches an expected specified behaviour. DTest is a sub-project of TSP (http ://savannah.nongnu.org/projects/tsp) and is used to test the CERTI project (http ://www.cert.fr/CERTI/index.fr.html) .

2 DTest structure

To define a distributed test sequence, we have to add steps to DTesters. A DTester is associated to a session handler allowing to manage a connection to one computer. Through this session handler, it can control the execution of processes mainly by monitoring its output with the expectFromCommand step..

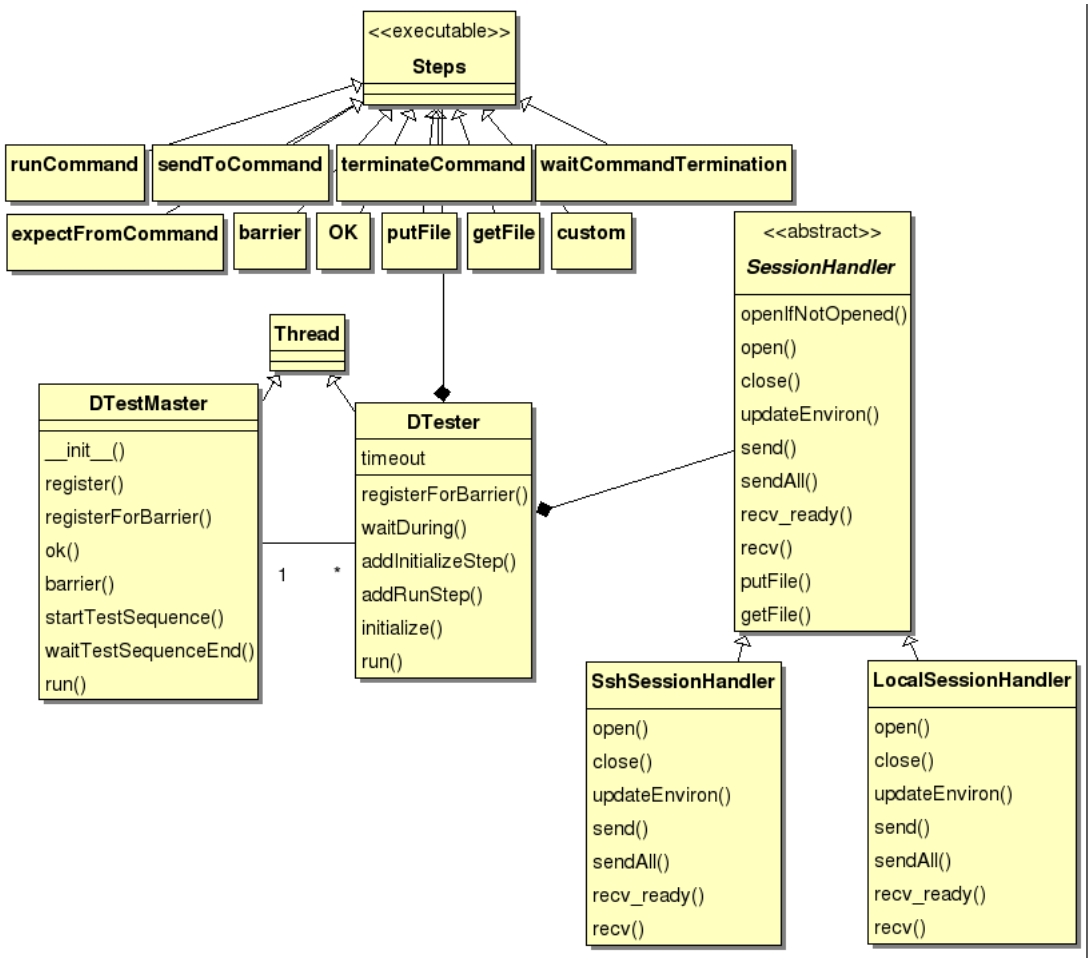
The DTesMaster coordinates all DTesters. It allows theirs synchronization through a barrier mecanism.

Here are listed the steps which can be added to a DTester :

ok Test Anything Protocol display method (http ://testanything.org/wiki/index.php/)

- *runCommand* runs a command on the computer through the ssh session
- *expectFromCommand* waits until the expected pattern is received on the session handler
- *terminateCommand* sends to the session handler the command for terminating
- *barrier* waits until every stakeholder have reached the specified barrier
- *sendToCommand* sends directly a string on the session handler
- *waitCommandTermination* waits until the session handler ends
- *custom* allows ro tun any python " callable "

Each DTester manages 3 files contening sendings, receipts and errors of the ssh session which can be used for debugging.



3 DTest Md5script example

The test sequence is defined by the user by adding execution steps to DTesters.

We will illustrate how DTest works with a simple file copy example. So here we want to assure that a file has been correctly copied from a server A to a server B. To do that, we want to compare the md5sum of the original file of A to the md5sum of the copied file of B we suppose that python is installed on A and B.

Here only one barrier (line 118 and 129) is used to assure that of course, we don't put source file on server B until we get it from server A.

The expectFromCommand steps (line 113 and 134) are used to wait that md5script writes "md5done" on its output. Once it has been written, we are sure that the associated md5sum_result file have been generated and we can get this result file.

The script scenario is :

1. put md5script on server A - line 109
2. generate md5sum_result of source file - line 112
3. get source file from server A - line 115
4. get md5sum_result from server A - line 117
5. put md5script on server B - line 127
6. put source file on server B (becoming dest file) - line 131

7. generate md5sum on server B - line 133
8. get md5sum_result from server B - line 136
9. compare md5sum_result from source file (res1) to md5sum_result of dest file (res2) - line 141

```
106 #tester1 steps
107 tester1.addRunStep("ok",True,skip="%s starting, source file : %s"%(tester1.name,file1))
108 #1/put md5script on server A
109 tester1.addRunStep("putFile",md5scriptname,getDir(source_param['fich'])+md5scriptname)
110 tester1.addRunStep("runCommand",command="chmod +x "+getDir(source_param['fich'])+md5scriptname)
111 #2/generate md5sum of source file
112 tester1.addRunStep("runCommand",command="%s"%(md5command1))
113 tester1.addRunStep("expectFromCommand",pattern="md5done")
114 #3/get source file from server A
115 tester1.addRunStep("getFile",source_param['fich'],"source_file")
116 #4/get md5sum from server A
117 tester1.addRunStep("getFile",resultfile1,"res1")
118 tester1.addRunStep("barrier","source getted")
119 tester1.addRunStep("terminateCommand")
120 tester1.addRunStep("waitCommandTermination")
121 #we check that diff result on standard output is like "same files"
122 tester1.addRunStep("ok",True,skip="%s has finished"%tester1.name)
123
124 #tester2 steps
125 tester2.addRunStep("ok",True,skip="%s starting, destination file : %s"%(tester2.name,file2))
126 #5/put md5script on server B
127 tester2.addRunStep("putFile",md5scriptname,getDir(dest_param['fich'])+md5scriptname)
128 tester2.addRunStep("runCommand",command="chmod +x "+getDir(dest_param['fich'])+md5scriptname)
129 tester2.addRunStep("barrier","source getted")
130 #6/put source file on server B (becoming dest file)
131 tester2.addRunStep("putFile","source_file",dest_param['fich'])
132 #7/generate md5sum on server B
133 tester2.addRunStep("runCommand",command="%s"%(md5command2))
134 tester2.addRunStep("expectFromCommand",pattern="md5done")
135 #8/get md5sum from server B
136 tester2.addRunStep("getFile",resultfile2,"res2")
137 tester2.addRunStep("terminateCommand")
138 tester2.addRunStep("waitCommandTermination")
139 #9/compare res1 and res2
140 def compareFiles(dtester):
141     dtester.ok(filecmp.cmp("res1","res2"),"Compare res1 with res2")
142 tester2.addRunStep(compareFiles)
143 tester2.nb_steps += 1
144
145 # Here begins the test
146 dtest.DTestMaster.logger.setLevel(level=logging.WARNING)
147 dtest.DTester.logger.setLevel(level=logging.WARNING)
148 dtest.SSHSessionHandler.logger.setLevel(level=logging.WARNING)
149
150 myDTestMaster = dtest.DTestMaster()
151 myDTestMaster.setTrace(1)
152 #register dtesters in DTestMaster
153 myDTestMaster.register(tester1)
154 myDTestMaster.register(tester2)
155 myDTestMaster.startTestSequence()
156 myDTestMaster.waitTestSequenceEnd()
```

4 A word about synchronization

The expectFromCommand DTester step allows to monitor an expected string on the DTester ssh session. It allows the tester to check a process behaviour by controlling its output. It is also used as a synchronization mean because, the DTester waits until the string is received or raises a timeout error.

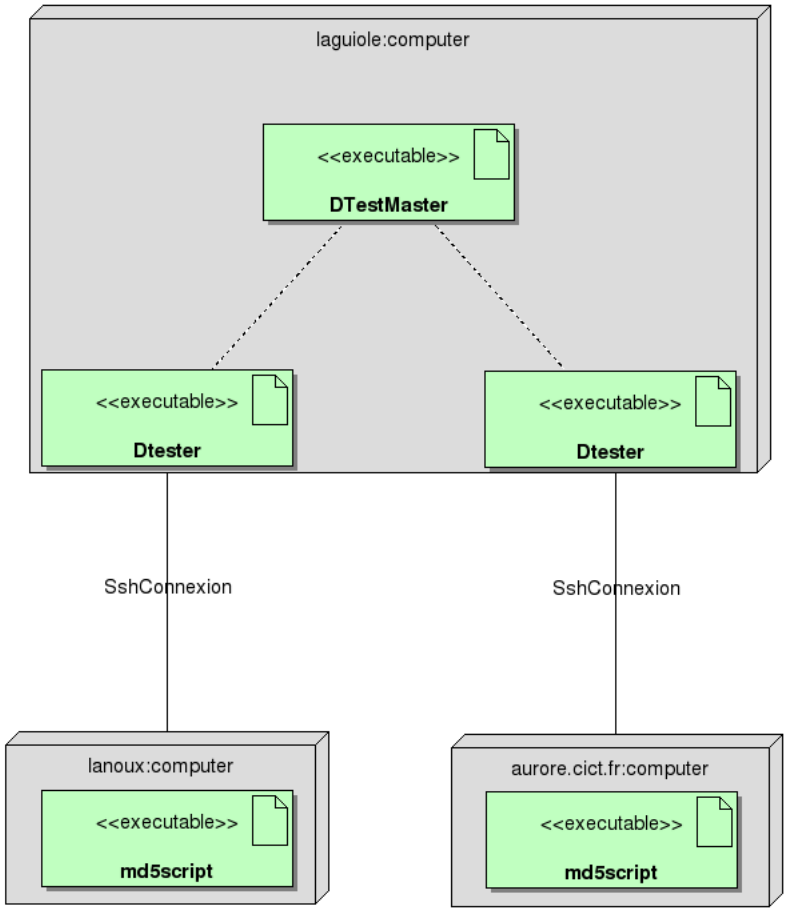
The barrier mecanism is used to synchronize different dtesters. A DTester reaching a barrier step "barrierA" is blocked until all DTesters participating to this barrier have reached their barrier step

”barrierA”.

We can use the waitDuring dtester method to locally wait before we run the next step.

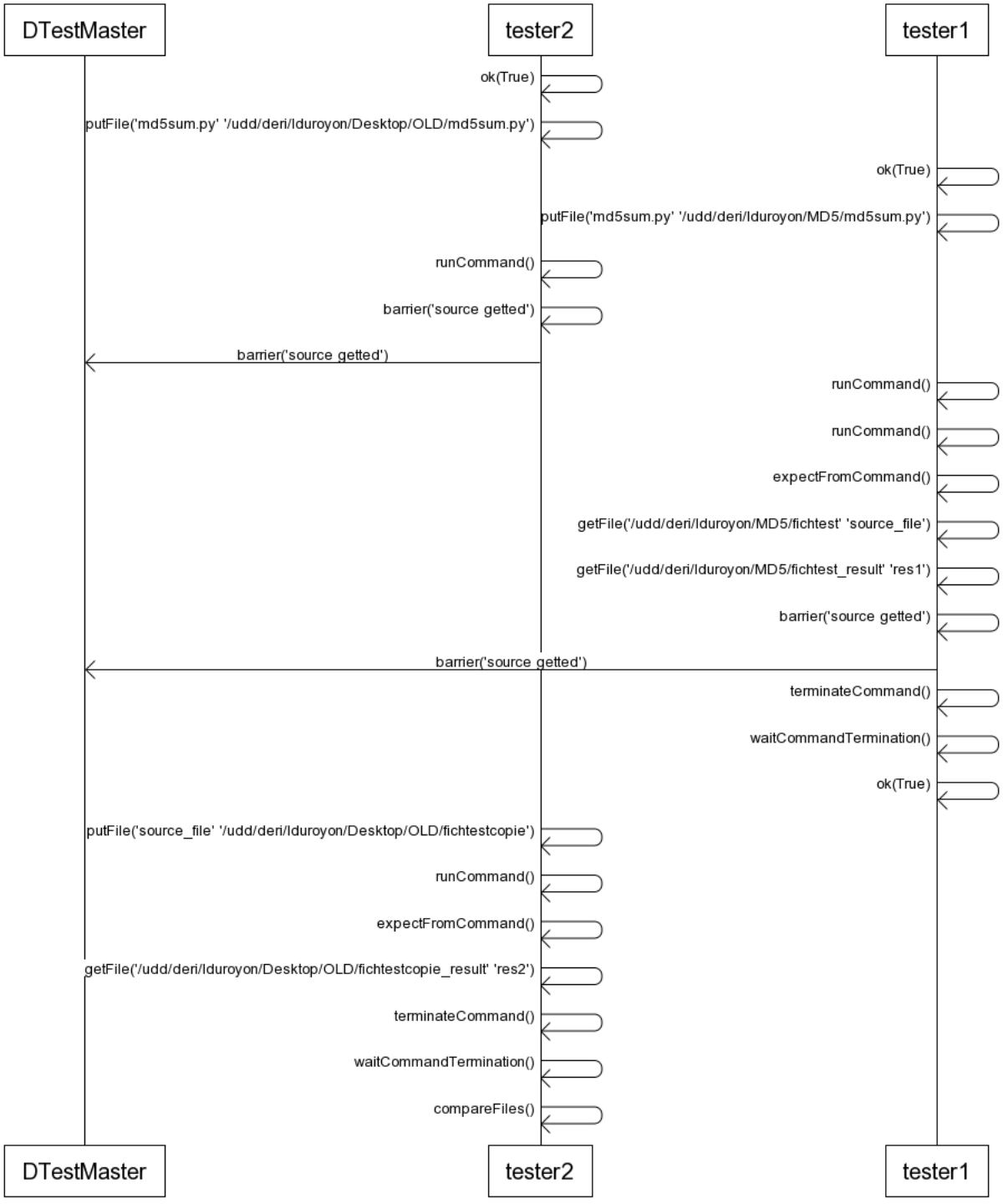
5 Md5script deployment diagram

The following diagram shows the DTest architecture for the md5 example. Here, tester1 manages a ssh connection to lanoux and tester2 manages a ssh connection to aurore.cict.fr. DTestmaster and DTesters are running locally.



6 Md5script msc diagram

This Message Sequence Chart diagram represents the execution trace of the md5script example. It is generated directly from its execution.



7 DTest required improvements

As DTest is still in development, it requires some improvments which are mainly :

1. handle X11 forwarding in order to be able to drive X11 application
2. handle ok/nok more automatically for every DTester operation method
3. add options for expectFromCommand to modify its semantics (actually the steps following expectFromCommand are always executed if expectFromCommand timeout or not)

4. improve monitoring for expectFromCommand : the way of monitoring is too simple and probably innefficient when the session output is huge

8 Outlook

Another goal of DTest is to verify properties about the execution of the distributed test sequence. This approch contrast with model driven engineering which intends to verify a model with a model checker at the specification step. Here properties are verified directly on execution traces (represented as MSC).

9 Useful links

CVS DTest repository : http ://cvs.savannah.nongnu.org/viewvc/dtest/ ?root=tsp
Message sequence charts : http ://www.etsi.org/WebSite/Technologies/MSC.aspx