

Testing GUI Java Applications with AGEDIS

by Alexander Herdt, imbus AG (alexander.herdt@imbus.de)

When testing the AGEDIS Test Execute Engine (TEE) we faced the challenge to develop a test environment with which it would be possible to execute tests automatically without additional tools. This involved starting the TEE with all ATS and TED files sequentially, running the tests and storing the results. In cases in which an API of the Java-classes of the TEE was not available, it was not possible to steer the software by calling internal routines. Instead there was a possibility to work on the TEE via the GUI-interface, which is also possible in Java. To avoid automating the test execution as an independent application, we used the TEE not only as SUT, but also as the active test execution engine, and built the following test environment:



Fig. 1 The general test environment.

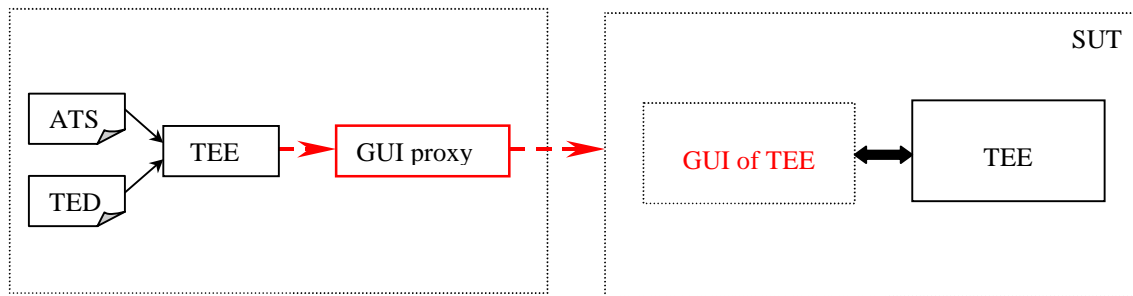


Fig. 2 The test environment when testing the TEE.

The TEE is steering the SUT (in this case another instance of the TEE), by accessing the SUT-GUI via a “GUI-proxy”. The GUI-proxy was implemented as a Java class and currently has only a small part of the functionality, which could be used during a GUI-test. It is however capable of handling the entire TEE-GUI.

Equipping the TEE with the GUI-proxy made it possible to push buttons on the SUT, to read its property values, to wait for a state of the SUT etc. The GUI-proxy is an interface offering the following functionality:

1. Converting the ATS-description of a GUI-element to a physical description
2. Translating the action description in Java Code
3. Searching for a GUI-window in the SUT
4. Searching for a GUI-element within a window
5. Executing a GUI-function at the SUT-element (pushing a button, filling in an **EditBox** etc.)

An ATS-file which is used for a GUI-test must identify a GUI-element unambiguously. The GUI-proxy uses a very simple identification mechanism because the GUI of the TEE is a simple one. A button, for example, can easily be identified by its label and the name of the “parent”-window. The ATS-description of a simulation or observation for pushing a button can look as follows:

```

<stimulate signature="doAction(string,string,string,string,string)" object="proxy">
  <params>
    <value>javax.swing.JButton</value>
    <value>AGEDIS Test Suite Driver</value>
    <value>Start</value>
    <value>doClick</value>
  </params>
</stimulate>
  
```

In this case, the GUI-proxy function

```
public void doAction(String class_name, String win_name, String label, String action, String param)
```

will look for an instance of the class **javax.swing.JButton** with the label “Start” within the window “AGEDIS Test Suite Driver”. When the button was found the GUI-proxy will call the method **doClick()** of the class mentioned above. The function **doAction()** is implemented as follows:

```
public void doAction(String class_name, String window_name, String text, String action, String param){
    Object[] params;
    Class[] classes;
    Component component = null;

    if (param.trim().equals("")) {
        params = null;
        classes = null;
    } else {
        params = parseParameters(param);
        classes = parseClasses(param);
    }

    component = searchComponent(class_name, window_name, null, text);
    try{
        if (component != null){
            Method method = Class.forName(class_name.trim()).getMethod(action.trim(),classes);
            method.invoke(component, params);
        } else {
            System.out.println("Component labeled \""+text+"\" not found!");
        }
    } catch
    ... exception handling.
```

The function will find the instance of the class **class_name**, which corresponds to the parameters **window_name** and **text**. Afterwards it calls the method **action()** for the instance.

As you can see, it is also possible to use the function for instances of other classes (e.g. for **JMenuItem** or **TextField**). As parameter of **action()** the names of the function of the class **class_name** can be used. The parameter **param** contains the types and values of the parameters for the handover to the calling function. Currently the GUI proxy is supporting the types int, String and boolean for handover. The value for the parameter **param** for the function **TextField.setText(String text)** could be described in the ATS-file as follows:

```
...
<value>string::new text for edit field</value>
...
```

Single steps of the GUI proxy search for a GUI window and element:

1. Finding all frames and dialogues of the application under test. With this step it is important to realize that not only visible windows can be found. Also a window can be presented in several instances which are distinguished only by their order. Furthermore, the process must be carried out recursively, since one frame can have several “child”-frames or “child”-dialogues.
2. Finding all “child”-elements of a window. Here, every element can act as container for other elements. This function must therefore also be carried out recursively.
3. Testing the class of a found element. If the class is identical with the specified one, the other characteristics like “name”, “label”, “size” etc. must be compared.

How do you know, which characteristics of an element have which values and which of those values can be used for an unambiguous description? A list of all GUI-elements of the SUT can be generated using a simple Java class. A prototype of the so called GUI-explorer was implemented for the TEE-test. It is the task of the

AGEDIS

www.agedis.de

GUI-explorer, to search all GUI-elements of the SUT (the SUT must run at the time) and to record the characteristics and their values. It is also possible to select the characteristics used for identification automatically.

The described technique was designed and partially implemented for a Java graphical user interface, but it is possible to test all other GUIs with this technique. The GUI-proxy can be implemented for different platforms.