Animator test plan

DaCoPAn

Helsinki 8th May 2004

Software Engineering Project

UNIVERSITY OF HELSINKI
Department of Computer Science

UNIVERSITY OF PETROZAVODSK
Department of Computer Science

Course

581260 Software Engineering Project (6 cr)

Project Group

Carlos Arrastia Aparicio
Jari Aarniala
Alejandro Fernandez Rey
Vesa Vainio
Jarkko Laine
Jonathan Brown

Kirill Kulakov
Andrey Salo
Andrey Ananin
Mikhail Kryshen
Viktor Surikov

Customer

Markku Kojo

Project Masters

Juha Taina (Supervisor)
Yury Bogoyavlenskiy (Supervisor)

Turjo Tuohiniemi (Instructor)
Dmitry Korzun (Instructor)

Homepage

`http://www.cs.helsinki.fi/group/dacopan`

Change Log

Version	Date	Modifications
1.0	07.05.2004	First version

Contents

1 Introduction **1**

2 Unit testing **1**

3 Integration testing **2**

3.1 Integration of internal components of Animator 2

3.2 TC001 - ProtocolEventsReader and Dataview 2

3.3 TC002 - Settings and user interface visualization. 3

3.4 TC003 - MSC buttons and encapsulation 4

3.5 TC004 - Localization 5

3.6 TC005 - Help menu 5

3.7 TC006 - Scenario and explore modes 5

3.8 TC007 - Time panel 6

3.9 TC008 - Note panel 6

3.10 Integration of DaCoPAn Analyzer and Animator modules 6

3.11 TC009 - Manually generated PEFs compliance to the XML DTD 7

3.12 TC010 - Analyzer generated PEFs compliance to the XML DTD 7

4 System testing **7**

4.1 TC011 - Loading a given tcpdump log pair into DaCoPAn software 8

4.2 TC012 - Loading a networking scenario tcpdump log pair into DaCoPAn software 8

5 Validation testing **9**

5.1 TC013 - User requirements 9

5.2 TC014 - Use cases 9

5.3 TC015 - System requirements 10

1 Introduction

The DaCoPAn Animator test plan document is intended to present several aspects of the testing to be performed on the implemented Animator module. It also contains a section dealing with the integration testing between Analyzer and Animator DaCoPAn modules, and another section which explains the system testing to be accomplished on the whole system.

Section 2 presents the unit test approach for the Animator module.

Section 3 takes care of integration testing both internally to the Animator module and between Analyzer and Animator modules. Internally to the Animator module, the testing of the interfaces between components (Java classes or Java interfaces) will be explained. Integration testing between the two main DaCoPAn modules will test the interface between them, that is that the connected Analyzer output and Animator input are as much error-free in their interaction as possible.

Section 4 presents system testing that will try to identify defects that will only surface when the complete system is assembled, in a black-box test approach.

The compliance to the requirements presented in the DaCoPAn requirements specification document will be checked in section 5.

The format chosen to present the test cases is:

- Each test case will be presented in a subsection to make it as readable as possible.
- Each test case will be given an identifier.
- Each test case will define its objective, and/or a textual description if needed.
- The input for the test case and/or the steps to follow to complete it will be specified as well.
- The expected output or behaviour of the software will be included.

2 Unit testing

This section contains an explanation of the internal unit testing approach of the Dacopan Animator module, using JUnit white box oriented tests.

The DaCoPAn animator unit tests are using JUnit. JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks. Most of the tests on the different Java classes extend from junit.framework.TestCase. More information about this class can be found in www.junit.org/junit/javadoc/3.8.1/index.htm or junit.sourceforge.net/javadoc/ It is possible to generate random data that will be used in test cases. One objective of using JUnit is that it makes possible to run several tests at once. Failures are detected and highlighted, and there's no need to check every single result.

Given that the chosen language for the Animator implementation is Java, which is object-oriented, test cases should check consitency in the use of attributes, check the behaviour of the objects in all their possible states, and finally test all operations associated with the object. Object interaction testing will be presented in integration testing among components of the Animator module in a further section.

Documentation on created tests is available in HTML format generated with Javadoc in the address: [add link when the documentation is posted]. The Javadoc documentation is created by adding comments in a specific format for each Java class, interface, field and method created. The HTML version for those Javadocs is created and published using an ANT target called publish-javadoc invoked from the directory containing the documented test cases.

3 Integration testing

3.1 Integration of internal components of Animator

This section presents test cases that check the interaction of the different Animator modules, after each of them has passed its correspondant unit test.

3.2 TC001 - ProtocolEventsReader and Dataview

- OBJECTIVE
  Test local integration between ProtocolEventsReader and Dataview: Most of the possible errors in Animator could come from the reading of the protocol interchange data from PEFs into the animator data structures. Errors in that scope should show error messages to the user, and reject erroneous PEFs. This test case should check the integration between DataView and ProtocolEventsReader. The loaded PEF must be a correct DTD compliant document, then the information contained in it must be consistent. Tests that are completed:
  - The system must resolve the path to the XML DTD.
  - The document must be valid (correct XML version and such.)
  - The DOCTYPE specified in the PEF should be correct.
  - Hosts, flows, links, layers, variables and units must be present. Protocols must be present inside layer tags.
  - The information contents must be read without errors into the data structures.
  - The protocol exchange contents must be consistent: each unit should have an end tag (received or lost), variables used in units should have been declared in the variables area and constants should have an "owner" (link, host or protocol).

- INPUT/ STEPS
  Run XMLProtocolEventsReaderTestCase.java as a JUnit test.

- EXPECTED OUTPUT/ BEHAVIOUR
  No errors or failures should be collected.

3.3 TC002 - Settings and user interface visualization.

- OBJECTIVE
  Test local integration between settings and user interface visualization.

- INPUT/ STEPS
  Make changes in the settings: change refresh delay, time scale and visual scale, select visualization of different variables and protocol header fields in the different layers. For that, run Animator module, load a correct PEF using the "open" command, then press settings button or press File->Settings in the upper menu bar.

  - 1 - Reduce refresh delay and press play.
  - 2 - Increase refresh delay and press play.
  - 3 - Reduce time scale and press play.
  - 4 - Increase time scale and press play.
  - 5 - Reduce visual scale and press play.
  - 6 - Increase visual scale and press play.
  - 7 - Select some variables and/or protocol header fields for each layer. You can change layer by pressing on the correspondant tab on the settings popup window. Clicking on the available variables and protocols header fields will select or deselect them.

- EXPECTED OUTPUT/ BEHAVIOUR
  The visualization should behave accordingly to the changes made in the settings:

  - 1 - Each step is smaller, so the visual flow of the animation is smoother.
  - 2 - Each step is larger, so the visual flow of the animation is less smooth.
  - 3 - Animation time goes faster.
  - 4 - Animation time goes slower.
  - 5 - The angle of the arrow representing a unit is smaller, so more units are represented on the screen.
  - 6 - The angle of the arrow representing a unit is larger, so less units are represented on the screen.
  - 7 - When visualizing each layer, the values of the protocol header fields of that layer should be represented on the arrows of the transfer units. Variables values are represented on columns on the sides of the unit transfers, symmetrically on both host sides.

3.4 TC003 - MSC buttons and encapsulation

- OBJECTIVE
  Test MSC buttons, and test encapsulation invocation.
- INPUT/ STEPS
  Run ExampleUITest.java. Random data is loaded into the Animator data structures and can be visualized. It is also possible to run Animator module and load a valid PEF to execute the test.
  - 1 - Press Play.
  - 2 - Press Pause.
  - 3 - Press Rewind.
  - 4 - Press Fast forward.
  - 5 - Press Rewind and then press Step forward.
  - 6 - Press Rewind and then press Step backward.
  - 7 - Press Layer selection buttons.
  - 8 - Press Show encapsulation button (in Unit flow area) for a given transfer unit (when enabled).
- EXPECTED OUTPUT/ BEHAVIOUR
  - 1 - Packet interchange should begin.
  - 2 - Packet interchange should be paused.
  - 3 - Packet interchange should rewind to the beginning.
  - 4 - Packet interchange should fast forward to the end.
  - 5 - Packet interchange should step once forward.
  - 6 - Packet interchange should step once backward.
  - 7 - Shown layer should change in MSC.
  - 8 - ENC panel should appear instead of MSC panel, showing the encapsulation tree for the chosen transfer unit. It shows units that are encapsulated inside the chosen unit if there are any and as well units from upper layers in which the chosen unit is encapsulated, if any. The chosen transfer unit should be highlighted with a red line. A partial unit means that not all the units that are encapsulated in underlying layers are shown. A left or right sibling of a unit on the left or right part of the shown units means that there are other previous or later units in the same layer.

3.5 TC004 - Localization

- OBJECTIVE
  Testing localization in the different animation features: That is testing that the program messages are correctly adapted to the language selected by the user.
- INPUT/ STEPS
  Run Animator module, and select a language in Language, in the upper menu bar. Run different test cases such as TC002 and TC003.
- EXPECTED OUTPUT/ BEHAVIOUR
  All the animation features should change their messages to the selected language. Error messages and Help menu should also be available in the selected language.

3.6 TC005 - Help menu

- OBJECTIVE
  Test help menu invocation.
- INPUT/ STEPS
  Run Animator module, and press Help button in the upper menu bar.
- EXPECTED OUTPUT/ BEHAVIOUR
  Help menu should be shown, offering different kinds of explanations about the use of the DaCoPAn Animator software.

3.7 TC006 - Scenario and explore modes

- OBJECTIVE
  Test switching between scenario and explore modes.
- INPUT/ STEPS
  Run Animator module, and load an animation sequence file SCE. Use play button, pause button and switch between explore mode and scenario mode using the menu bar radio buttons Animation->Scenario mode and Animation->Explore mode.
- EXPECTED OUTPUT/ BEHAVIOUR
  A sequence of different presentation types will be shown as a playlist. In Scenario mode it should be possible to visualize a sequence of animations, change the scenario playlist and switch to explore mode, where the user should be able to explore the scenario using the different controls as described in other scenarios.

3.8 TC007 - Time panel

- OBJECTIVE
  Test time panel visualization.
- INPUT/ STEPS
  Run Animator module, load a PEF and press play or step forward.
- EXPECTED OUTPUT/ BEHAVIOUR
  The time panel should show the actual time of the animation, that is the timestamp corresponding to the position of the nowline shown in the MSC animation panel.

3.9 TC008 - Note panel

- OBJECTIVE
  Test note panel.
- INPUT/ STEPS
  Run Animator module and load a valid PEF. Edit some text on the note panel, and press Save note button.
  - 1 - Press Rewind and Play button.
  - 2 - When a note is shown, press Delete note, then press Rewind and then Play buttons.
- EXPECTED OUTPUT/ BEHAVIOUR
  - 1 - The note should be saved and later shown for the same layer and at the same timestamp. When the same timestamp is reached during animation, the note must be shown.
  - 2 - At the timestamp when the old note was, no note should be shown.

3.10 Integration of DaCoPAn Analyzer and Animator modules

Integration between DaCoPAn Analyzer and Animator modules relies essentially on the format and contents of the Protocol Events Files generated by the Analyzer and read by the Animator. The compliance of any PEF to the DTD is checked in the XMLProtocolEventsReader class, and written according to it with the pefwrite.c file functions. Some control of the consistency of the data is also achieved in XMLProtocolEventsReader class, that may reject the use of any incomplete or contradicting information contained in any PEF.

3.11 TC009 - Manually generated PEFs compliance to the XML DTD

- OBJECTIVE
  Test PEF compliance to the XML DTD with manually generated PEFs: The integration of the Analyzer and the Animator modules relies on the DTD compliance of the PEF that is passed from Analyzer to Animator. The information present in the PEF should be correctly loaded into the data structures. So, test number TC001 can be considered as a global integration test as it checks the reading of the PEF into the Animator.

- INPUT/ STEPS
  Run XMLProtocolEventsReaderTestCase.java as a JUnit test.

- EXPECTED OUTPUT/ BEHAVIOUR
  No errors or failures should be collected.

3.12 TC010 - Analyzer generated PEFs compliance to the XML DTD

- OBJECTIVE
  Test PEF compliance to the XML DTD with Analyzer generated PEFs: ProtocolEventsReader should be tested with some PEFs created by the Analyzer module. PEF is written using the pefwrite.c functions and read using the ProtocolEventsReader Java file methods. All local tests (unit and local integration tests) completed on pefwrite.c functions and ProtocolEventsReader Java file methods are testing the actual integration between the Analyzer and Animator modules. The information present in the PEF should be correctly loaded into the data structures. Tests should be similar to those made in test case TC001.

- INPUT/ STEPS

  - Run Animator module and load the Analyzer generated PEF.
  - Visually, compare the visualized data interchange information with the information present in the PEF.

- EXPECTED OUTPUT/ BEHAVIOUR
  No errors or failures should be collected on loading, meaning that the loaded PEF conforms to the XML DTD. The packet interchange data must be the same than the one contained in the PEF created by the Analyzer module.

4 System testing

This section presents some test cases to be completed on the DaCoPAn software when Analyzer and Animator modules will be assembled, in a black-box testing approach. The

visualization of the protocol exchange data must correspond to the information present in the tcpdump log pair.

4.1 TC011 - Loading a given tcpdump log pair into DaCoPAn software

- OBJECTIVE
  Test loading any given tcpdump log pair into DaCoPAn software.
- INPUT/ STEPS
  - Run Analyzer module with a tcpdump logs pair.
  - Run Animator module and load the Analyzer generated PEF.
  - Visually, compare the animated data interchange information with the information present in the tcpdump logs.
- EXPECTED OUTPUT/ BEHAVIOUR
  The visualized information must correspond to the the one present in the tcpdump logs pair.

4.2 TC012 - Loading a networking scenario tcpdump log pair into DaCoPAn software

- OBJECTIVE
  Test loading networking scenarios 201, 310, 321, 341, 361, 362 (priority 1 networking scenarios) tcpdump logs into DaCoPAn software.
- INPUT/ STEPS
  - Run Analyzer module with tcpdump logs pair correspondant to the desired networking scenario.
  - Run Animator module and load the Analyzer generated PEF.
  - Check that visualized information corresponds to the networking scenario description in the DaCoPAn Networking scenarios document.
- EXPECTED OUTPUT/ BEHAVIOUR
  The DaCoPAn Networking scenarios document provides a description of the expected behaviour in these scenarios.

5 Validation testing

This section proves through tests how the Animator requirements presented in the DaCoPAn Requirements specification document are satisfied by the implemented solution.

5.1 TC013 - User requirements

- OBJECTIVE
  Test that user requirements for Animator are satisfied. All user requirements are presented in the DaCoPAn Requirements specification document.
- INPUT/ STEPS
  Run Animator module and load a valid PEF using the load button.
  - 1 - MSC animation.
  - 2 - Press Show encapsulation button (in Unit flow area) for a given transfer unit (when enabled).
  - 3 - Press MSC control buttons, layer buttons and settings button.
  - 4 - Edit notes and press Save note button. Press Delete button.
- EXPECTED OUTPUT/ BEHAVIOUR
  - 1 - MSC animation should be visible by default.
  - 2 - ENC panel should appear instead of MSC panel, showing the ENC tree for the chosen transfer unit.
  - 3 - It should be possible to control and tune the Animation, and the protocol data to be shown should be selectable.
  - 4 - Notes should be visualized in the appropriate layer and timestamp, and can be deleted.

5.2 TC014 - Use cases

- OBJECTIVE
  Test use cases for Animator. All use cases are presented in the DaCoPAn Requirements specification document.
- INPUT/ STEPS
  - 1 - Use case 5.2.2: Run Animator module and load a PEF using the load button.

  - 2 - Use case 5.2.3: Run Animator module and load a PEF using the load button. Then press play and pause buttons to control animation.
  - 3 - Use case 5.2.4: Run Animator module and load a PEF using the load button. Then press step forward and backward buttons to control animation. Press settings button to change the number of steps completed each time the step buttons are pressed.
  - 4 - Use case 5.2.6: Run Animator module and load a PEF using the load button. Press play. Notes can be added and deleted to MSC and ENC animation by editing them in the Note panel and pressing Save note button.
  - 5 - Use case 5.2.7: Run Animator module and load a PEF using the load button. Press Settings button and select changes.
  - 6 - Use case 5.2.8: Run Animator module and load a PEF using the load button. Press Settings button and select changes. Press the Save file or the Save scenario button if enabled and choose a file where changes will be saved.

- EXPECTED OUTPUT/ BEHAVIOUR

  - 1 - If the file is not rejected, MSC animation should be visible.
  - 2 - MSC animation should be played and paused accordingly.
  - 3 - MSC animation should step forward and backward.
  - 4 - Notes should be visible when at the same time and layer on later visualizations, unless they were deleted.
  - 5 - Visualization should change accordingly to the changes.
  - 6 - It should be possible to Run Animator again and load the scenario file. The animation settings should be the same than those that were saved.

5.3 TC015 - System requirements

- OBJECTIVE
  Test system requirements for Animator. All system requirements are presented in the DaCoPAn Requirements specification document. Since most of them specify concrete user interface visualization features, we can consider that local integration tests (section 3) concerning visualization matters prove that those requirements are satisfied.

- INPUT/ STEPS
  Do test cases from TC002 to TC008.

- EXPECTED OUTPUT/ BEHAVIOUR
  The expected outputs and/or behaviours for these tests are presented in section 3.