Implementation: Animator

DaCoPAn

Helsinki 31st 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	20.05.2004	First version

Contents

1 Introduction **1**

2 Data structures **1**

2.1 The Link class 1

2.2 The StepIterator interface 1

2.3 The notes framework 1

3 Animations **1**

3.1 Encapsulation animation 1

3.2 Message Sequence Chart 2

3.3 Unit Flow Orchestration 2

3.4 Note panel 3

4 PEF reader **3**

4.1 The ProtocolEventsReader interface 3

4.2 The ProgressIndicator interface 3

5 Scenario data **3**

5.1 The Saveable interface 3

5.2 The ScenarioFile class 3

5.3 The ObjectSerializer interface 3

6 Scenario play lists **4**

7 Localization **4**

7.1 The Localization class 4

8 Main user interface **4**

8.1 Menu and tool bar command actions 4

1 Introduction

This document summarizes the changes made to the design of the Animator during the implementation phase. These design changes are listed here and also reflected in the Design document of the Animator. Any designed features not implemented are also discussed.

Each higher-level module of the Animator is discussed in a separate chapter of this document.

2 Data structures

2.1 The Link class

When the data model of the protocol events data was refined further, the need for a class representing a physical link between two hosts in a network arose. Thus, the `Link` class was introduced.

2.2 The StepIterator interface

A new interface that encapsulates the stepping functionality in an animation.

2.3 The notes framework

The notes framework of the Animator was designed to be defined in an interface called `NotesInterface`. A more descriptive name, `NoteManager`, was used in the implementation. Many of the methods of NoteManager were renamed. The names were updated in the UML diagrams as well as in the textual content of the notes framework chapter.

The different note types, ENC notes and MSC notes, were eventually implemented in a single class, `Note`, instead of two separate classes, as the design suggests.

3 Animations

3.1 Encapsulation animation

The creation of the encapsulation tree of a certain unit proved out to be more complex than it seemed in the design phase. The logic and algorithms involved in creating such a tree were implemented and refactored into a separate class in the `model` -package called `EncTreeModel`.

Due to time constraints, some planned features of the encapsulation animation were not implemented, but their initial design can be found in the Design document. These features are:

- The encapsulation animation is not exactly an animation at the moment: instead, it's more of a static diagram. Animating the encapsulation tree can be implemented later.
- Sizes of units and payload are not specified in the diagram in any way.
- Layer names are not visualized.
- There are no special symbols for empty layers.

3.2 Message Sequence Chart

The Message Sequence Chart animation was implemented mostly as designed.

One major feature of the MSC panel that was not designed was the ability to move the current time ("now line") by clicking or dragging the mouse inside the MSC panel. This feature was implemented.

The "draw below now line" feature exists in the code, but currently there is no way to turn it on in the user interface.

The column model for MSC panel contains more complex logic than was mentioned in the design. The logic is related to dividing the available space across different column. In the case that the variable columns are able to get all the width they need, the extra space is given to the drawing area. If not, the drawing area is kept at a pre-defined minimum width and the remaining space is divided to the columns according to their relative required widths.

The values are drawn in columns so that if they fit inside the column, they are aligned to right. However, if the whole value doesn't fit, the value is aligned to left so that the beginning of the value will be visible.

The drawing of the MSC panel uses an optimization technique in play mode. Because of this, certain values that normally would appear below the now line (note icons, column values for send events) will be clipped at the now line. However, this only happens when the now line is moving, so in normal use it is very difficult to notice the difference. The optimization improves the performance of drawing very much.

3.3 Unit Flow Orchestration

Unit Flow panel was implemented otherwise as designed, except that functionality for manually selecting the units was not implemented.

3.4 Note panel

The note panel was not designed in a detailed way, but it was eventually implemented. The note panel is considered an animation panel, because it tracks the same time as other animation panels. The note panel always shows the note that is mapped to a point in time that is equal to current time or the any succeeding point in time before a time where the next note on the same layer is mapped.

4 PEF reader

4.1 The ProtocolEventsReader interface

The `ProtocolEventsReader` interface's only method, `read()`, was refactored a bit: the reader now accepts a `java.io.Reader` instead of an input stream and it accepts a `ProgressIndicator` as a third parameter.

4.2 The ProgressIndicator interface

To show visible progress to users when loading PEF files, a simple progress indicator interface was added to the `ui` -package for this purpose.

5 Scenario data

5.1 The Saveable interface

When moving into the implementation phase, the design for the classes responsible of saving the scenario data was improved. An interface called `Saveable` was introduced into the `scenario` -package. This interface and an explanation of its purpose were added to the Design document.

5.2 The ScenarioFile class

Changed method signatures to reflect the introduction of the `Saveable` interface, updated UML diagrams.

5.3 The ObjectSerializer interface

Added a method (`setDataView(view:DataView):void`) for setting the DataView instance that's used when loading scenario files and resolving references to model objects.

6 Scenario play lists

The scenario play list was implemented mostly as designed. The user interface for the list, with buttons for editing, is currently in its own non-modal dialog. It is possible to add items to any point in the list, edit settings for MSC items, and delete items from the list. The recording functionality contains some automatic recording actions.

The textual breakpoints were not implemented. They are partially replaces by pause items that cause the scenario to wait for user input, but don't contain any text, or don't show a dialog.

7 Localization

7.1 The Localization class

Although briefly described in the Design document, the localization mechanism of the Animator was not designed on the class level. In the implementation phase the central class for the localization mechanism (`Localization`) was designed and implemented, and these changed are reflected in the Design document.

8 Main user interface

8.1 Menu and tool bar command actions

When the animation sequence design was refined during implementation phase it was decided that the different modes should be kept as little different from MainFrame's point of view as possible. Because of this, the commands for controlling animation (play, pause, rewind, fast forward) do no distinction between the animation mode.

Also removed the record command from the menu, as recording is handled by the animation sequence and not MainFrame.