Giraffe:
Graphical Front-ends for 2D Animation

Nicholas Daley
Masters Thesis
The University of Waikato
New Zealand
Email: ntdaley@acm.org
Supervisor - Dr Mark Utting

2003

A knowl gm nts

ark Utting • Supervisor.

etra alik

Cont nts

L st of F gur s

Chapter 1

Introduction

Basic research is what I'm doing when I don't know what I'm doing.
– Werner Von Braun

This thesis describes the creation of the Caffe package as part of the ZT classes or code that appeared in a Gd e interface would either have to be compiled into the Gd e enimator, or be imported in a dynamic library. W

C pt r 5

ff Ar t tur

.1 Gaffe Interfaces

art of the plan for Gaffe interfaces was that they would look much like any application. Because of this, interfaces need to be capable of being quite complex, with components of many types potentially arranged into panels and sub-panels. It was also desired that designers of Gaffe interfaces should not need to write significant amounts of program code to achieve their goals. Thirdly, Gaffe should place as few restrictions as possible on how an interface appears; e.

.2 ni ation

The way the Gaffe animator code handles interfaces has been dealt with above. Other than this, the main decisions with animation were:

1. How to manage the history of animation states.

4. If a bean should be able to trigger actions in other objects ('events'), then it should provide methods for registering listeners. When the appropriate conditions happen to trigger such an event, the bean calls the corresponding method on all of the registered listeners of appropriate type, passing an event object. e.g. [illegible] bean that fires an event when one of its properties change would have a method for registering the event:

```
  public void addPropertyChangeListener
                                   (PropertyChangeListener [illegible]
                                   public void [illegible]
```

4.2 Introspection

`java.beans.Introspector` allows the `BeanInfo` object for a class to be obtained; this can be done with any type even non-beans. ■

C

BSFEngine must be available, and regi[illegible]

apply(...) method of a ZLocator to retrieve the value it wants.

6.2.2 The User Interface ortion

This section describes the second of the two portions (shown in Figure 6.3).

ost of the work in the user interface portion is done by the beans in the interface loaded from the file (labeled as 'Bean' in the U L diagram in Figure 6.3). Other than this, the most interesting parts are For , ni ator ore, and History.

For represents one window in the interface. It also tracks all of its descendant beans using a Bean ontextServices; to which the For itself is added as a service (See section 4.4), so that beans at eavly acces th f om t ey f om. I p ovides me thds f o ad ig andr movi g bea s (t e by the des ign r , and by t e fil f ormat[5]), methods

if there is a problem with the input, then the animator engine will throw an exception.

The classes `BSFService rovider` and `HistoryService rovider` are used

6. Designer

Java Pa kage: `net.sourceforge.cz`

Form

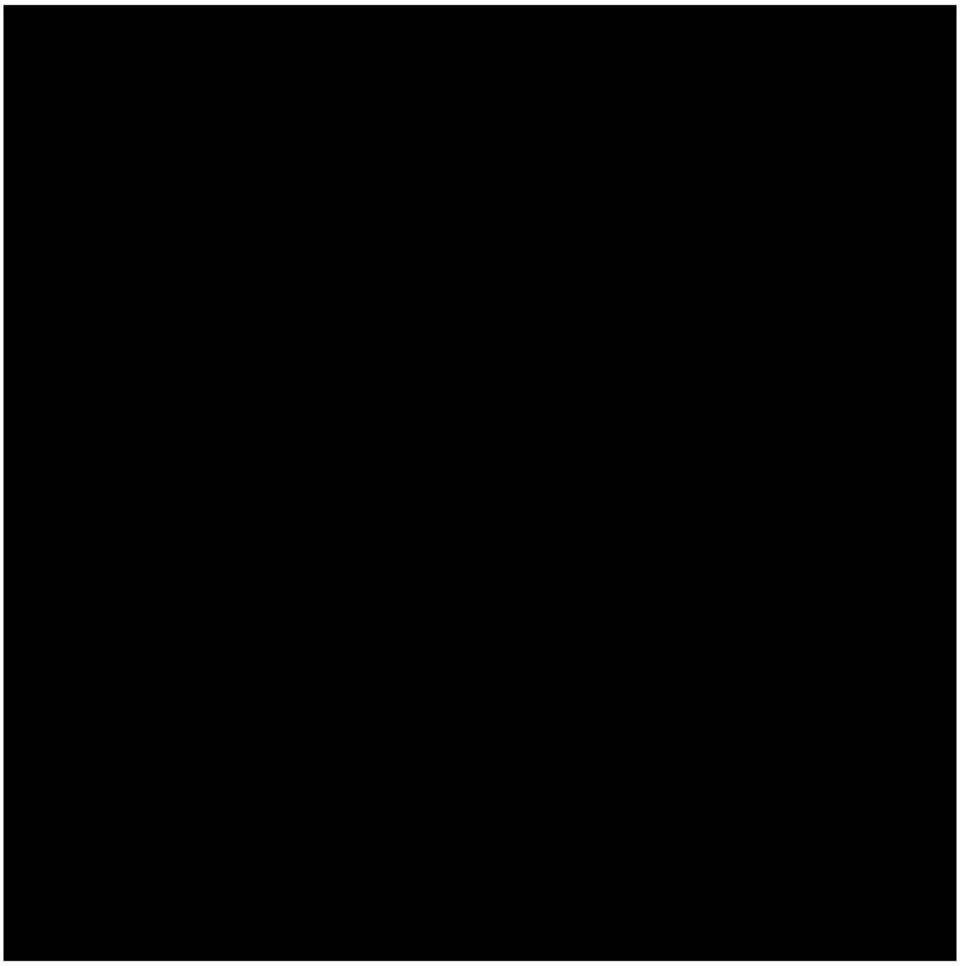


Figure . : BirthdayBook's Add input form being edited.

6. .1 Tools

Every tool provides:

- (optionally) an icon to display in its button in the tool window.
- A name to display if there is no icon.
-

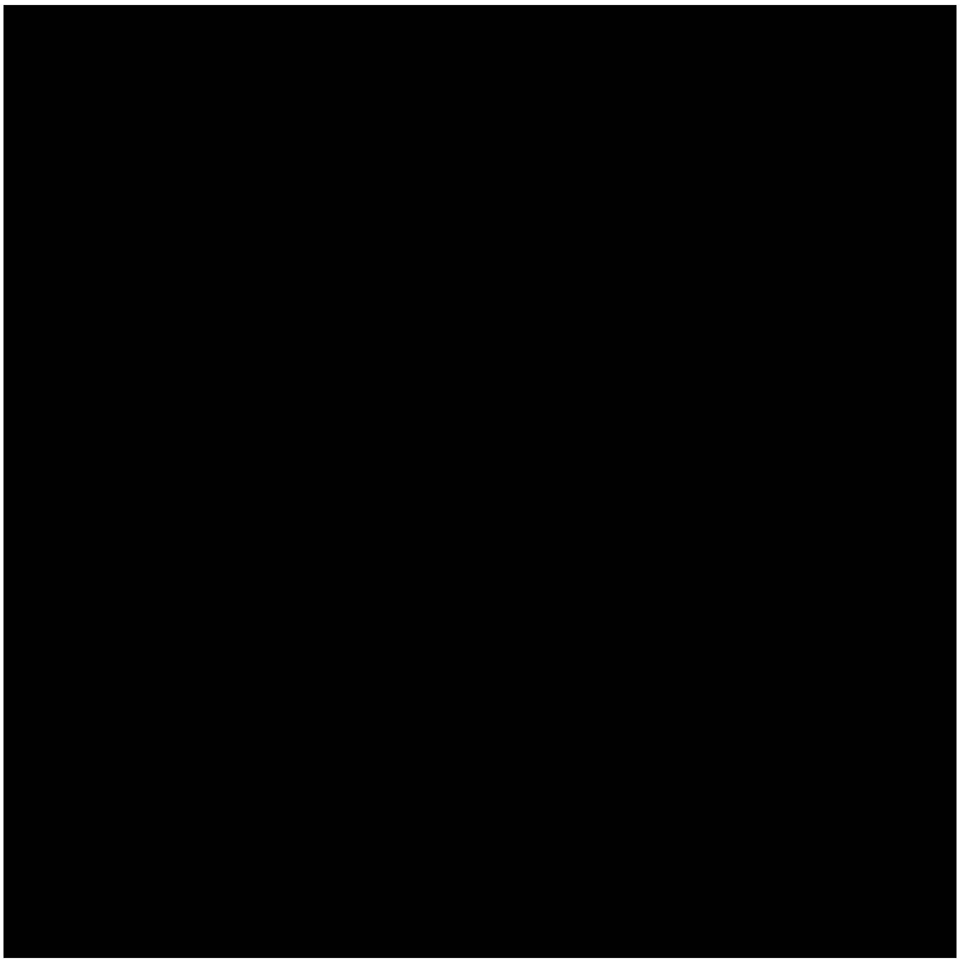


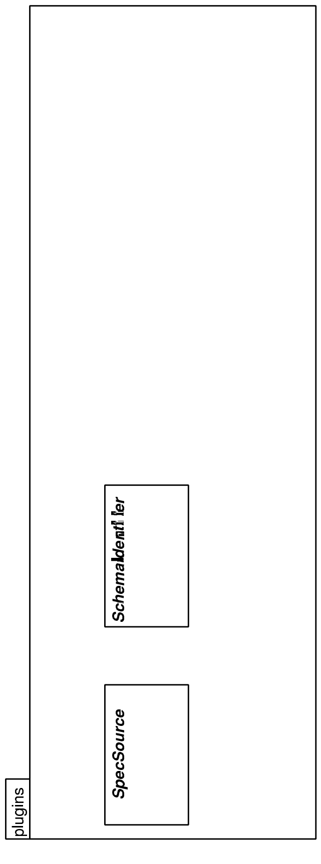
Figure . : The roperties Window.

C pt r 7

D s gn of t n r tor

The main program only accesses the plug-ins through a `luginList`, which keeps track of plug-ins, instantiates plug-in implementations, and handles most of the option processing.

After option processing, the main program goes through each plug-in in turn, feeding it the data it may need from previous plug-ins, and extracting the data later plug-ins may need.



```
luginList'
```

The method `obtainSpec` is used to get the parsed specification, throwing an
e

.2. Schema Identifier

Interfa e:

```
net...plugins.Scne aIdentifier
```

```
public interface SchemaIdentifier extends Plugin {
  public static final String optionName="identifier";
  public static final String name="Schema Identifier";
  public void identifySchemas(Term specification,
                              List schemas)
    throws IllegalStateException;
  public  onstDec   getStateSchema();
  public  onstDec   getInitSchema();
  public List getOperationSchemas();
};
```

The method `identifyScne as` takes the specification from `SpecSource`, and the list ofon frst ofon e n p o s

Default Implementation:

`net...plugins.impl.BasicBeanInterfaceGenerator`

Example Interface Screenshots - BirthdayBook

These screenshots are from the Gaffe animator, using an interface generated from the BirthdayBook example Z specification in Appendix [illegible].

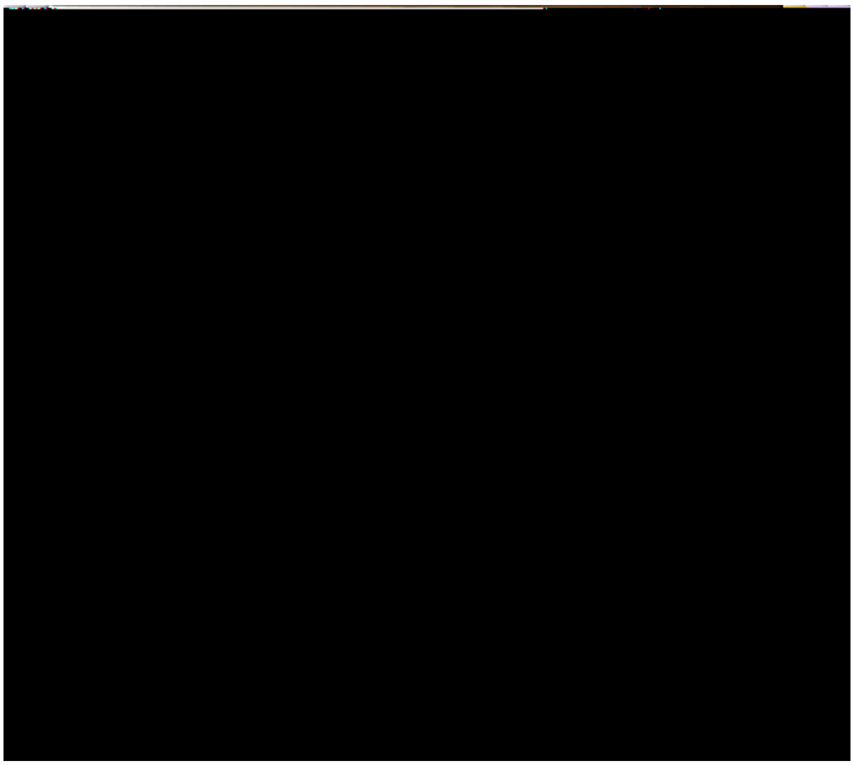


Figure 7.2: The state window of the generated interface.

Note that, although the birthday variable is a relation, it has appeared as a text field rather than as a table. This is because, the Gaffe generator is not yet very smart about determining some variable types. Once the CZT type-checker is written, Gaffe will be able to use it to determine variable types, and the generator will correctly produce a two column table.

The variables are updated by a script function (`fillBeans`) that is called by a script which is triggered by a `History Proxy`[4]; this function matches Z variables to components based on the name property of the component, then does what is needed, depending on the component's type, to display the variable through the component. The first row of buttons contains one for each operation, each of which trigger scripts that open the appropria[illegible]

The bottom row of buttons contains buttons to step back and forth through the history, and back and forth through the current set of solutions [5]; also it contains labels to display the current position in the history. The scripts associated with these buttons just call the appropriate method on the `History` object, and the labels are updated by a `History roxy`.

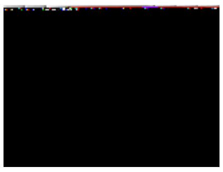


Figure 7.3: The AddBirthday input window.

Input windows display their variables in much the same w

Figure 7. : The Remind input w

.2.6 Variable Extractor

Inte fa e:

```
  net...plugins.variableExtractor


public interface VariableExtractor extends Plugin {
  public static final String optionName="variable";
  public static final String name="V
```

V

in the Gaffe animator on its bottom pane, and a glass pane that handles user interaction (and displays handles for resizing, event link highlighting, etc.) on its top pane, the designer can show .

Because the location property for `Forms` is only used by the designer, sav-

and this is the behaviour used by the current JavaScript engine! The only

.6 Fle ible configuration

Some way was needed to make addition of tools, bean types, and property editors achievable without recompiling. It was a simple matter to add an initialis ation script to the designer, allowing these and other settings to be configured.

.7 No back-end

Because the animator engine that Gaffe attaches to isn't written yet, all testing has been done with custom `History` implementations that fake the back-end for a particular specification.

10.4 GUI

10.7 llow for s to ave enus

t present the designer des not allow for menus and menu bars. Beca xe menus are significantly different from normal components, this would probably mean a separate editor ithe Gaffe designer for handling menus. This could be handled as a property editor in the properties w

App n x A

rt y ook Ex mpl

.1 T7

The operation schemas:

[illegible]ddBirthday

$known, known' : \mathbb{P}$ [illegible] E

[illegible] E [illegible] day [illegible] E

Robust versions of the operations:

$$REPORT ::= ok \mid already\_known \mid not\_known$$

Success

$$result! : REPORT$$

$$result! = ok$$

AlreadyKnown

$$known, known' : \mathbb{P}\, NAME$$
$$birthday, birthday' : NAME \nrightarrow DATE$$
$$name? : NAME$$
$$result! : REPORT$$

$$known = \operatorname{dom} birthday$$
$$known' = \operatorname{dom} birthday'$$

```
                                      = (ZGiven) inputs_.get(ZLocator.fromString("name?"));
                                    final ZGiven dateInput
                                      = (ZGiven) inputs_.get(ZLocator.fromString("date?"));
                                    System.err.println("+++++" + nameInput + "\t" + dateInput);
                                    if (currentKnown.nput);
ut);
```

```
D [illegible] Z [illegible] fromString("date?")); [illegible] Z System.err.println(" [illegible] +++++ [illegible] nameIn
```

```
= (ZGiven) inputs_.get(ZLocator.fromString("name?"));
```

```
    newResultsM.put("date!", dateOutput);
  } else {
```

App n x

An E rly U