GUISurfer as a starting point for CROSS Task

João Carlos Silva João Saraiva José Creissac Campos

Department of Computer Science

CROSS Café
November 11, 2009

Members of the research team task 3

- João Alexandre Saraiva;
- José Creissac Campos;
- João Carlos Silva;
- Carlos Silva;
- Rui Gonçalo.

Projects

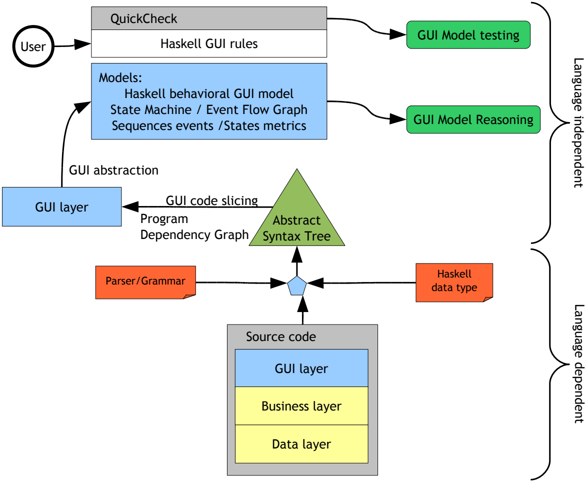
- IVY: A model-based usability analysis environment (FCT-funded project POSC/EIA/56646/2004) which aimed at developing a model based tool for the analysis of interactive systems designs.
- GUIsurfer, a generic tool to reverse engineer GUI code.
- CROSS Task T3: Graphical User Interface Analysis. This task is to develop techniques and tools that will enable analysis of the user interface layer of software systems from source code.

Motivation

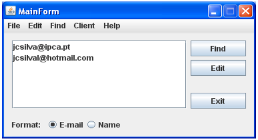
To improve the productivity of the programmers, there are tools that allow for the fast development of user interfaces.
However:

- The code defining the GUI is a mix of programmer and tool generated code
- The code produced by such tools is difficult to understand and manipulate.
- The tools do not provide support for GUI reasoning and testing.

The GUIsurfer Architecture

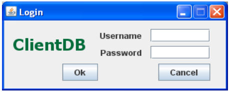


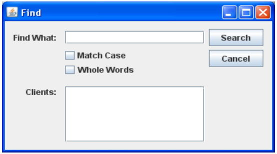
An Interactive Agenda



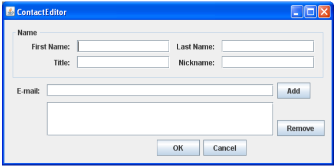
jcsilva@ipca.pt
jcsilval@hotmail.com

Format: ● E-mail ○ Name





□ Match Case
□ Whole Words

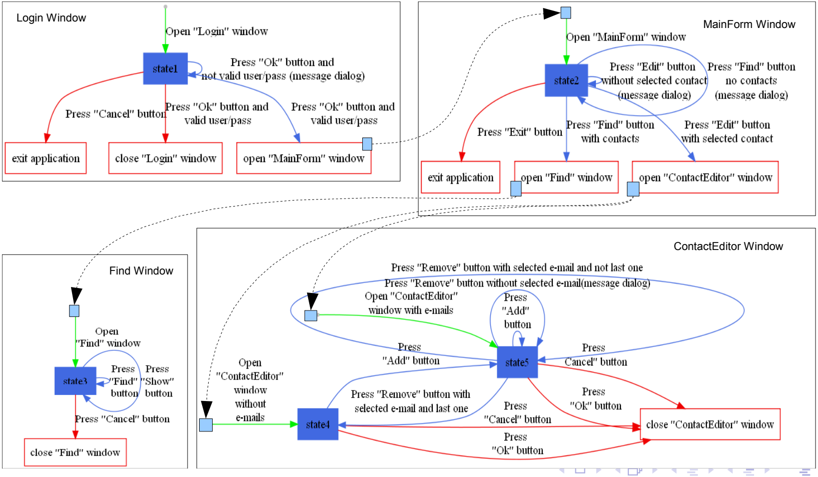


An Interactive Agenda

The behaviour of the GUI is described in a programming language by defining functions/methods which associate events to interactive actions. For example, in Java/swing the action performed when the `Ok` button is pressed is as follows:

```
private void OkActionPerformed(...)
{if (isValid(user.getText(),pass.getText())==true)
 {new MainForm().setVisible(true);
  this.dispose();}
 else javax.swing.JOptionPane.showMessageDialog
          (this,"User/Pass not valid","Login",0);
}
```

A GUI Behavioral Model



Login Window

Open "Login" window

state1

Press "Ok" button and not valid user/pass (message dialog)

Press "Cancel" button

Press "Ok" button and valid user/pass

Press "Ok" button and valid user/pass

exit application

close "Login" window

open "MainForm" window

MainForm Window

Open "MainForm" window

state2

Press "Edit" button without selected contact (message dialog)

Press "Find" button no contacts (message dialog)

Press "Exit" button

Press "Find" button with contacts

Press "Edit" button with selected contact

exit application

open "Find" window

open "ContactEditor" window

Find Window

Open "Find" window

state3

Press "Find" button

Press "Show" button

Press "Cancel" button

close "Find" window

ContactEditor Window

Press "Remove" button with selected e-mail and not last one

Press "Remove" button without selected e-mail(message dialog)

Open "ContactEditor" window with e-mails

Press "Add" button

Press "Add" button

state5

Press Cancel" button

Open "ContactEditor" window without e-mails

Press "Remove" button with selected e-mail and last one

Press "Cancel" button

Press "Ok" button

state4

Press "Ok" button

close "ContactEditor" window

From a GUI Behavioral Model

Having defined the GUI of the Agenda via a static machine, we are able to use techniques to reason about and test the application:

- we can compute an equivalent machine with the minimum number of states (refactoring).
- we can use graph algorithms, to detect if all states are reachable from the initial one, in order to detect whether a particular window will ever be displayed or not (dead code elimination).
- Finally, we can generate valid (and non-valid) sentences of the language defined by the machine, to be used as test cases in order to prove properties of the interface (testing).

The Goal of the GUIsurfer tool

- To develop a tool to automatically extract models containing GUI behaviors: when a GUI event can occur, which are the related conditions, which interactive actions are executed and which GUI states are generated.
- To be able to reason about GUI models in order to analyse the application's usability, and the quality of the implementation.
- To define generic techniques so that we can analyze interactive applications written in different programming languages.

GUI Reverse Engineering

In order to manipulate the GUI of the interface we need to *focus* our techniques in the part/aspect of the source code that defines the interface. Thus, we use two generic techniques

- **Strategic programming**: in order to traverse any abstract syntax tree (AST) and focus our attention in the constructors of the visual objects and actions.
- **Code slicing**: We use standard slicing techniques to compute a program dependency graph and extract the interface aspect from the source code.

GUI Reverse Engineering

To define the GUI slicing code, we look any widget that enables: *user input*, *user selection*, *user action* or *output to user*.

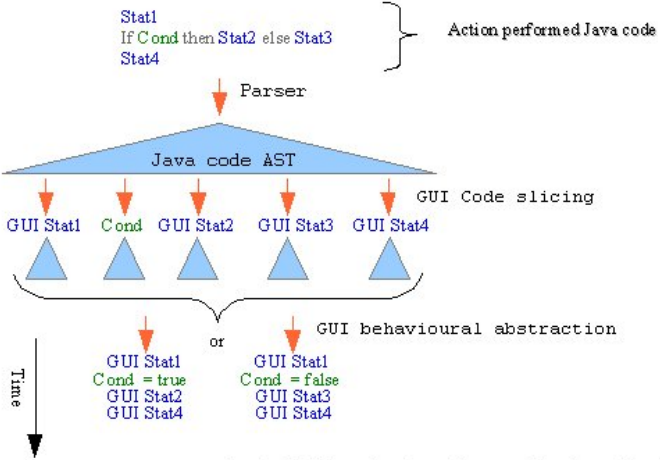
As an example, to extract all buttons definitions from a *java/swing* AST we can execute the following instruction:

```
selection javaAST ``JButton'' 1 1
```

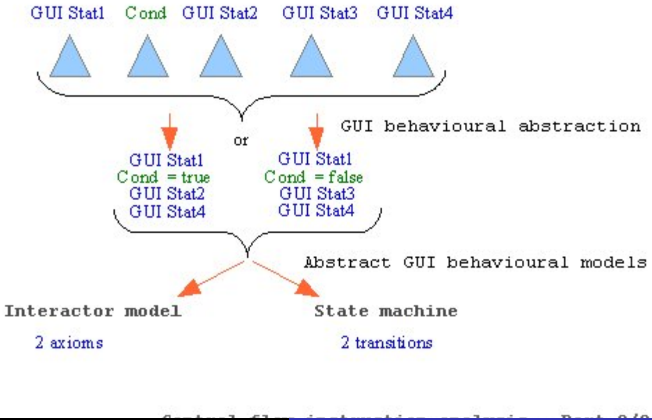
From a WxHaskell AST, the same action could be executed as:

```
selection wxHaskellAST ``button'' 1 1
```

GUI Reverse Engineering - Control Flow



GUI Reverse Engineering - Control Flow



GUI Stat1 Cond GUI Stat2 GUI Stat3 GUI Stat4

GUI behavioural abstraction

GUI Stat1
Cond = true
GUI Stat2
GUI Stat4

or

GUI Stat1
Cond = false
GUI Stat3
GUI Stat4

Abstract GUI behavioural models

Interactor model

2 axioms

State machine

2 transitions

Control flow instruction analysis - Part 2/2

GUIsurfer tool - Example of use

- FileParser Login.java
- AstAnalyser Login.java.ast main JButton,setEnabled,exit,showMessageDialog,dispose
- Graph eventsFromInitState.gui initState.gui 0 windowName.gui Login ClientDBjava 1

GUI Models - Haskell Model

An algebraic data type defining a generic GUI behavioral model of the interface:

```
type GuiModel = Map (EventRef,CondRef) [ExpRef]

type EventRef = String
type CondRef = String

type Pres = Map ExpRef (EventRef,Bool)
type End = [ExpRef]
type Close = [ExpRef]
type NewWindow = Map ExpRef WindowName

type WindowName = String
type ExpRef = Int
```

GUI Models - Haskell Model - Login Window

As example, after slicing the *Login* window from agenda application we obtain automatically:

```
guimodel :: GuiModel
guimodel = fromList [(("Cancel","cond1"),[1])
                    ,(("Ok","cond2"),[2,3])
                    ,(("Ok","cond3"),[4])
                    ,(("init","condInit1"),[5,6,7,8,9])]
pres :: Pres
pres = fromList [(8,("Cancel",True)),(9,("Ok",True))]
end :: End
end = [1]
newWindow :: NewWindow
newWindow = fromList [(2,"MainForm"),(5,"Login")]
```

GUI Testing

Having expressed the slicing and modelling techniques in Haskell, we can now use *QuickCheck*: a *haskell* library tool for testing *Haskell* programs automatically.

- The programmer provides a specification of the program, in the form of properties, and QuickCheck tests the properties in a large number of randomly generated cases.
- Specifications are expressed in *Haskell*, using combinators defined in the QuickCheck library.

GUI Testing Example

We will test if the application satisfies the following rule: users can only access the following windows *Login*, *MainForm*, *Find*, *ContactEditor*.

Considering *lc* the sequence of valid events, we can specify the following property rule:

```
rule lc = fromList [a|(a,b,c)<-lc] ==
          fromList ["Login","MainForm",
                    "Find","ContactEditor"
                   ]
```

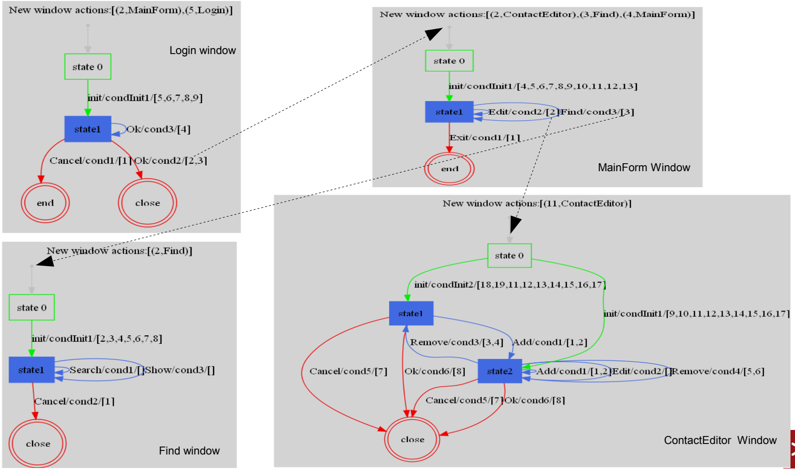
GUI Testing

Testing through *QuickCheck* the application's *GuiModel* with this rule , we obtain the following result:

```
OK, passed 10000 tests.
87\% events sequence length: 5.
11\% events sequence length: 4.
1.5\% events sequence length: 3.
0.5\% events sequence length: 2.
0\% events sequence length: 1.
```

The rule was tested in 10000 randomly generated cases. All of them satisfy the rule.

GUI Models - State Machine



Others Models

- Graph manipulations (intersection, difference)
- Deterministic finite automata manipulations (minimization, pattern matching)

Task 3: Objectives

- GUIsurfer back-end extension: generalizing the approach to new languages and toolkits (GWT, AJAX);
- GUIsurfer front-end extension: enabling the generation of new types of models in order to extend the analyses which can be performed (CTT);
- Extracted models analysis (patterns, metrics).

Sub-tasks

- João Carlos Silva (Phd thesis):
  - GUIsurfer extension;
  - Models analysis (Patterns, metrics).
- Carlos Silva (MSc thesis - CROSS / GUIA):
  - GWT / AJAX - Development of a back-end enabling GUISurfer to reverse engineer GWT / AJAX applications;
  - Concurrent Task Trees (notation for modelling and animation of hierarchical task models) - generation of CTT task models by GUISurfer.
- Rui Gonçalo (BII - Utilização de GUISurfer na análise de aplicações interactivas em Java/Swing):
  - Interactive applications repository;
  - GUIsurfer and similar tools manipulation.

GUISurfer as a starting point for CROSS Task

João Carlos Silva João Saraiva José Creissac Campos

Department of Computer Science

CROSS Café
November 11, 2009