Universidade do Minho
Escola de Engenharia
Departamento de Informática

Model-based User Interface Testing

IFIP WG 2.7/13.4 meeting on SE-HCI Processes

Amsterdam, 24-25 of November, 2007.

José Creissac Campos[(a)] with José L. Silva[(a)] & Ana C. R. Paiva[(b)]

[(a)]DI/CCTC, Universidade do Minho, Braga, Portugal
[(b)]FEUP, Universidade do Porto, Portugal

http://www.di.uminho.pt

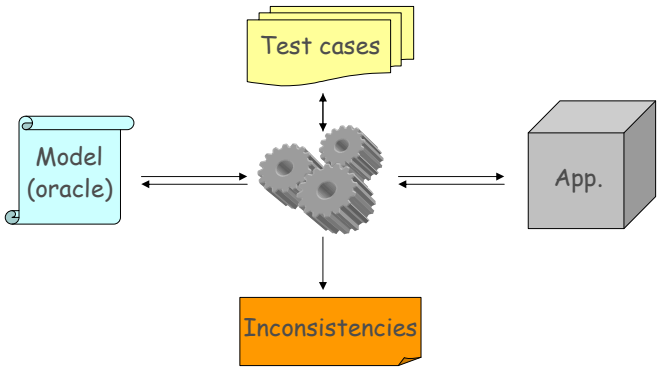
Motivation

- HCI and quality – focus is on the users
  - Analysis of device's UI design using models
  - Testing of device models/implementation with users
- SE and quality – focus is on the device
  - Analysis of device's (internal) architectural and behavioural models
  - Testing of implemented functionality
- Question: How to bridge the gap between HCI models and devices' implementations?
  - Correct by construction?
  - Testing?
- Model-based testing (MBT)
  - Testing (the implementation) against a model
  - We present a study on the use of task models for MBT

Talk Plan

- Model-based software testing (MBT)
- Model-based UI testing
- CTT task models
- A process for MBT with CTT and Spec Explorer
- Discussion
- Conclusions and Future Work

Model-based software testing



Model Based UI testing (I)

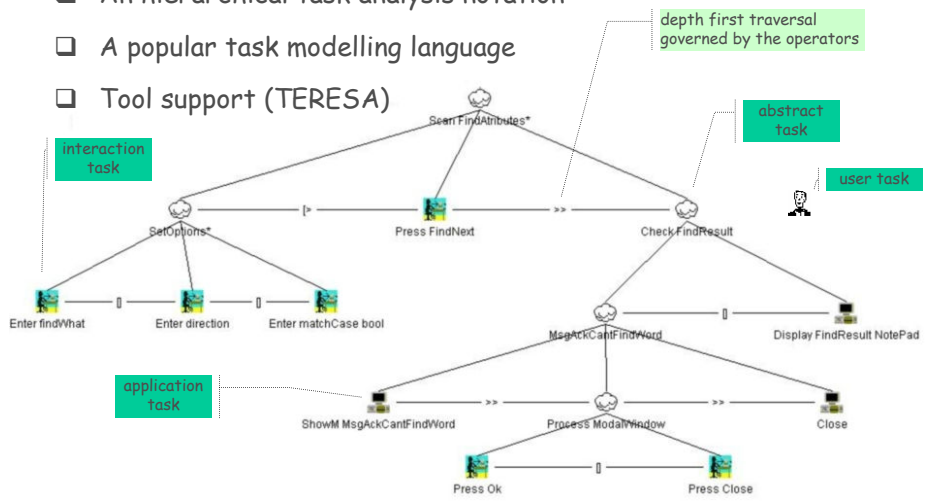
- Specific difficulties
  - Gap between UI model events and application UI events
  - Generation of UI events at application level
  - Need to consider complex behaviours (goals, tasks,…)
- Three challenges
  - Map concrete actions to/from abstract actions
  - Write adaptor code to simulate user actions
  - Need for adequate models of the UI

Model Based UI testing (II)

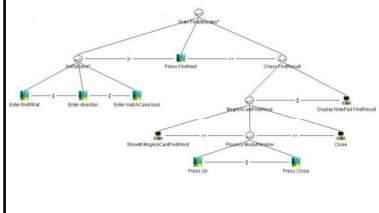
- Paiva (2007) has developed a model-based UI testing approach (around Spec Explorer / Spec#)
  - Guidelines to model GUIs in Spec#
  - GUI Mapping tool – automated mapping between the GUI model and its implementation
  - Strategies to avoid test case explosion
- However, Spec# not ideal to model GUIs
  - Too much like a programming language
  - Level of detail too low
- Need to find alternative modelling notations acknowledged
  - Task models for model-based testing?
  - Our conclusion: "Yes, but…"

CTT Task Models

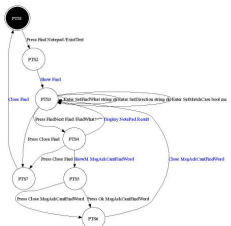
- An hierarchical task analysis notation
- A popular task modelling language
- Tool support (TERESA)



The Process



Task model (CTT)



FSM model (PTS)

```
namespace notepad2
{
using WindowManager;
using States;
using SaveDialog;
using FindDialog;
using OpenDialog;
using ReplaceDialog
[Action]
void Show_Notepad()
```

Oracle (Spec#)

(i)

(ii)
TERESA

(iii)
TOM

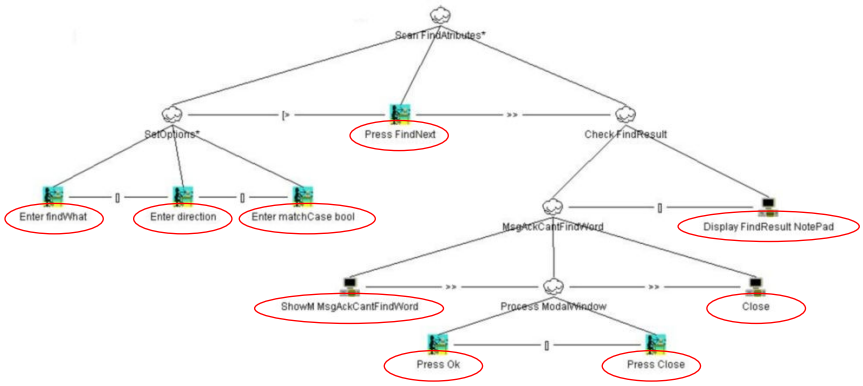
(iv)

(i) Using CTT models as oracles (I)

- Structuring the task models (FIT-based):
  - Start <task>
  - Enter <field> [<type>]
  - Press <button> [<window>]
  - Show <window>
  - ShowM <window>
  - Display <value> <window>
  - Close [<window>]
- Pre- and post-conditions on atomic tasks help add semantics to the model

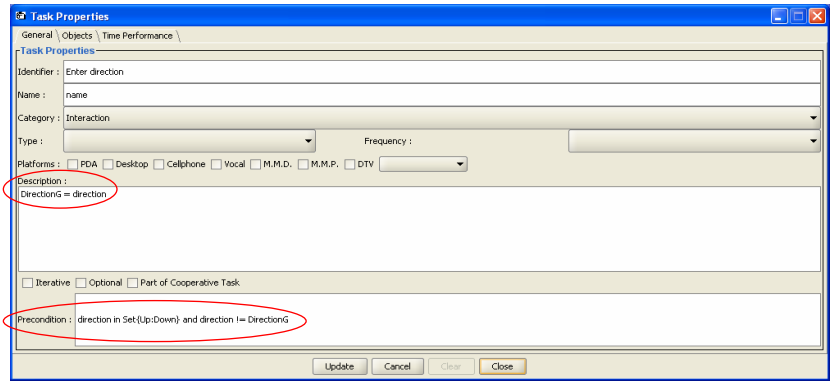
(i) Using CTT models as oracles (II)

- Using the keywords

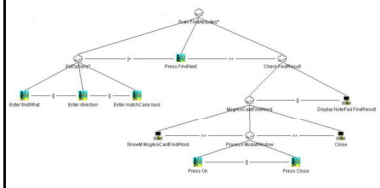


(i) Using CTT models as oracles (III)

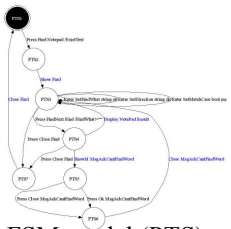
- Adding semantics to the model (Enter direction task)



The Process



Task model (CTT)



FSM model (PTS)

```
namespace notepad2
{
using WindowManager;
using States;
using SaveDialog;
using FindDialog;
using OpenDialog;
using ReplaceDialog
[Action]
void Show_Notepad()
```

Oracle (Spec#)

(i)

(ii)
TERESA

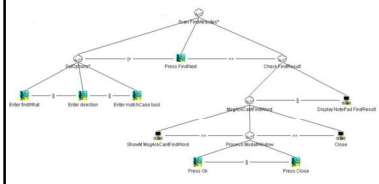
(iii)
TOM

(iv)

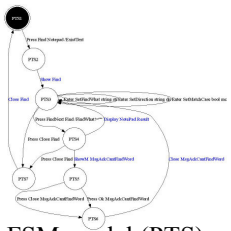
(ii) Generating a FSM representation

- Presentation Task Sets (PTS) are generated by TERESA
  - An automatic step, generates a XML file
- A few problems:
  - Technical: disabling ([>) and suspend/resume (|>)
    - Manual correction is used at the moment
  - Expressiveness: modal dialogues, wizards, …
    - Pre- and post- conditions used for modal windows
    - Do we want to model wizards in a task model?!

The Process



Task model (CTT)



FSM model (PTS)

```
namespace notepad2
{
using WindowManager;
using States;
using SaveDialog;
using FindDialog;
using OpenDialog;
using ReplaceDialog
[Action]
void Show_Notepad()
```

Oracle (Spec#)

(i)

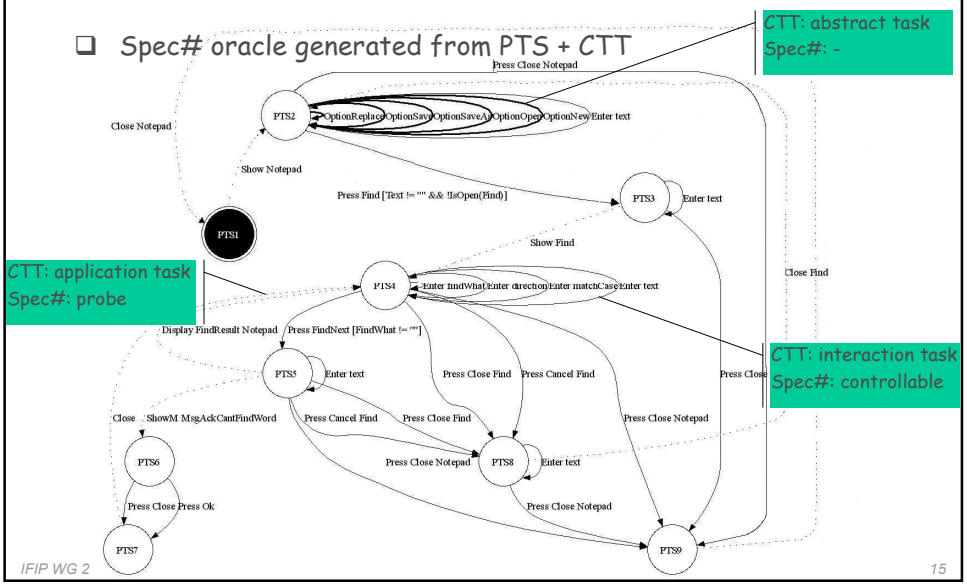
(ii)
TERESA

(iii)
TOM

(iv)

(iii) TOM – Generating the Spec# oracle (I)

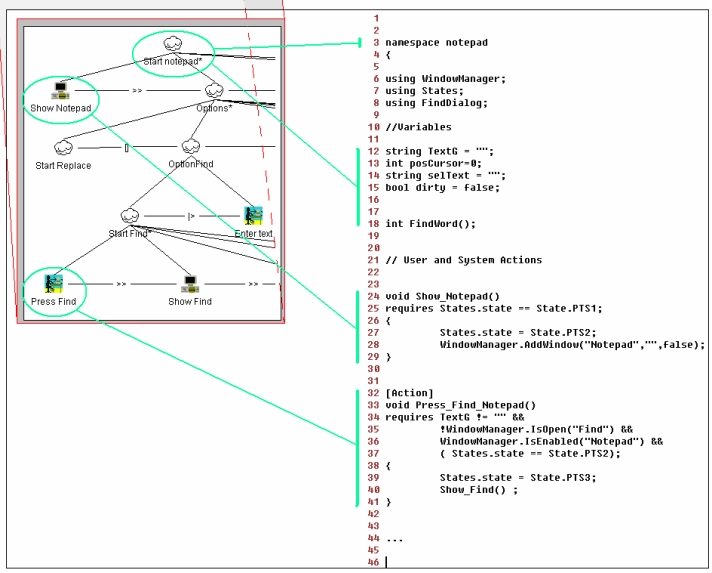
- Spec# oracle generated from PTS + CTT



(iii) TOM – Generating the Spec# oracle (II)

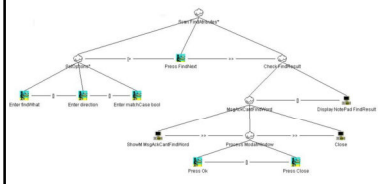
- Guidelines in (Paiva 2007) are followed
  - One module for each GUI window
  - State variables to describe window state/controls
  - Controllable methods to describe behaviour
  - Probe methods to observe state
- A window manager model is used to deal with windows
  - bool IsOpen(string name) {…
  - bool IsEnabled(string name) {…
  - void AddWindow(string name, string parent, bool isModal) {…
  - void RemoveWindow(string name) {…
  - …

(iii) TOM – Generating the Spec# oracle (III)

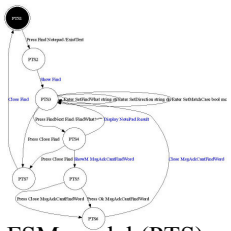



```
1
2
3  namespace notepad
4  {
5
6  using WindowManager;
7  using States;
8  using FindDialog;
9
10 //Variables
11
12 string TextG = "";
13 int posCursor=0;
14 string selText = "";
15 bool dirty = false;
16
17
18 int FindWord();
19
20
21 // User and System Actions
22
23
24 void Show_Notepad()
25 requires States.state == State.PTS1;
26 {
27         States.state = State.PTS2;
28         WindowManager.AddWindow("Notepad","",false);
29 }
30
31
32 [Action]
33 void Press_Find_Notepad()
34 requires TextG != "" &&
35         !WindowManager.IsOpen("Find") &&
36         WindowManager.IsEnabled("Notepad") &&
37         ( States.state == State.PTS2);
38 {
39         States.state = State.PTS3;
40         Show_Find() ;
41 }
42
43
44 ...
45
46
```


The Process



Task model (CTT)



FSM model (PTS)

```
namespace notepad2
{
using WindowManager;
using States;
using SaveDialog;
using FindDialog;
using OpenDialog;
using ReplaceDialog
[Action]
void Show_Notepad()
```

Oracle (Spec#)

(i)

(ii) TERESA

(iii) TOM

(iv)

(iv) Refining and using the oracle (I)

- Additional information is added to the model (manually)
  - Definition of Domains for input parameters (generalized MC/DC criterion)
  - Definition of additional behavior (e.g., FindWord())
  - Refinement of probe actions
- Glue code between model and implementation is created
  - Matching physical actions to oracle actions
  - Mapping between oracle and application

(iv) Refining and using the oracle (II)

- Test cases are generated
  - Spec Explorer does this automatically
  - Full transition coverage criterion was used
- The test cases are run
  - Spec Explorer does this automatically
  - The application is tested against the task model
- In this case, the application was not found not to conform to the task model

Discussion

- Models are at a level of abstraction familiar to user interface designers/developers
  - Models not at the level of detail needed for *typical* MBT
    - Tension between modelling how to use the system, and modelling the system
    - Annotations and pre-/post-conditions used to address this
  - Task models describe idealised user behaviour (hard to test for user error)
    - Use variations of the original task model
- The cost incurred in developing the oracle is much reduced
  - True!
  - Can be further reduced if convention is followed from the outset

Conclusion & Future work

- Despite limitations, using task models as oracles has proven viable
  - Care must be taken regarding what they are used for, and how
  - Interesting as a first validation of fitness for pourpose
  - Task models and MBT as a tool for component selection!
- Future work
  - Solving TERESA problems?
    - Still not clear what the problems are
  - Enriching task models with dialogue information?
    - This would allow testing of implementation details
  - Exploring fault injection into task models?
    - This will allow testing system response to user error

Thank you!

jose.campos@di.uminho.pt