MBT 2007
Third Workshop on Model-Based Testing
March 31 - April 1, 2007, Braga, Portugal

Towards the Integration of Visual and Formal Models for GUI Testing

Ana C. R. Paiva [1] (apaiva@fe.up.pt)
João C. P. Faria[1,2] (jpf@fe.up.pt)
Raul F. A. M. Vidal [1] (rmvidal@fe.up.pt)

Software Engineering Group
(http://www.fe.up.pt/softeng/)

(1)

(2)

Agenda

- Introduction
- Approach overview
- Modelling GUI structure
- Modelling GUI behaviour
- Protocol state machine translation
- State based test case generation and coverage analysis
- Modelling GUI usage for scenario based testing
- Conclusions and future work

Introduction: Previous work in MBGT

- Spec# (FSE/MSR)
  - Extends C# with contracts and high-level constructs
  - Can be used as modelling or programming language
- Spec Explorer (FSE/MSR)
  - Model based testing tool
  - Automates test case generation and execution for API conformance testing
  - Test sequences are derived from a FSM which is generated by bounded exploration of the (executable) model
- GUI testing extensions (Our previous work)
  - GUI modelling techniques promoting abstraction and reuse, to reduce the modelling effort
  - FSM reduction tool for hierarchical GUIs, to handle the state/test case explosion problem
  - GUI mapping tool, to handle the model-to-implementation gap

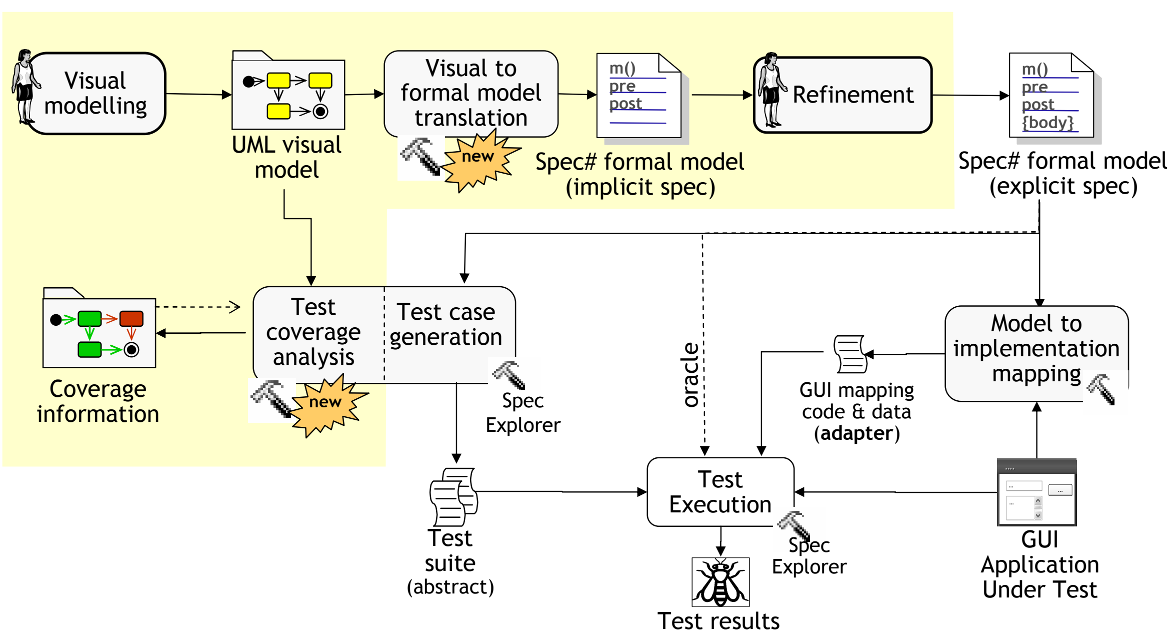
Introduction: MBGT Problems

- Modellers/testers don't like writing *textual* models, particularly for GUIs
  - UML models would be better accepted by testers, but in general, the detail and rigour of UML models is insufficient for being used as test oracles
- Lack of support of coverage criteria best adapted for GUI testing, such as coverage of navigation maps
  - With Spec Explorer, the bounded exploration process is based on parameter domain values provided by the tester, but there is no feedback mechanism to evaluate the quality of the test cases obtained based on appropriate test adequacy criteria for GUIs

Introduction: Aims

- Develop an approach for MBGT that combines the strengths of visual modelling (usability) and formal modelling notations (rigor)
- Provide a familiar visual modelling *front-end*, based on UML, on top of a formal modelling *back-end*, based on a formal specification language, such as Spec#
  - UML diagrams are partial views over the formal model
- Hide as much as possible formalism details from the modellers/testers
- Use the visual models as the basis for test adequacy/coverage criteria

Approach overview: Process



Approach overview: Model architecture

		UML (visual model/*front-end*)	Spec# (formal model/*back-end*)
Usage (optional)		▪ Use case diagrams ▪ Activity diagrams	▪ Scenario methods' pre/posts, bodies & assertions
GUI	Structure	▪ Class diagrams	▪ Classes/modules, methods, state variables, invariant'
	Behaviour	▪ Protocol state machine diagrams ▪ ---	▪ Action methods' pre/posts ▪ Method bodies
Domain layer			

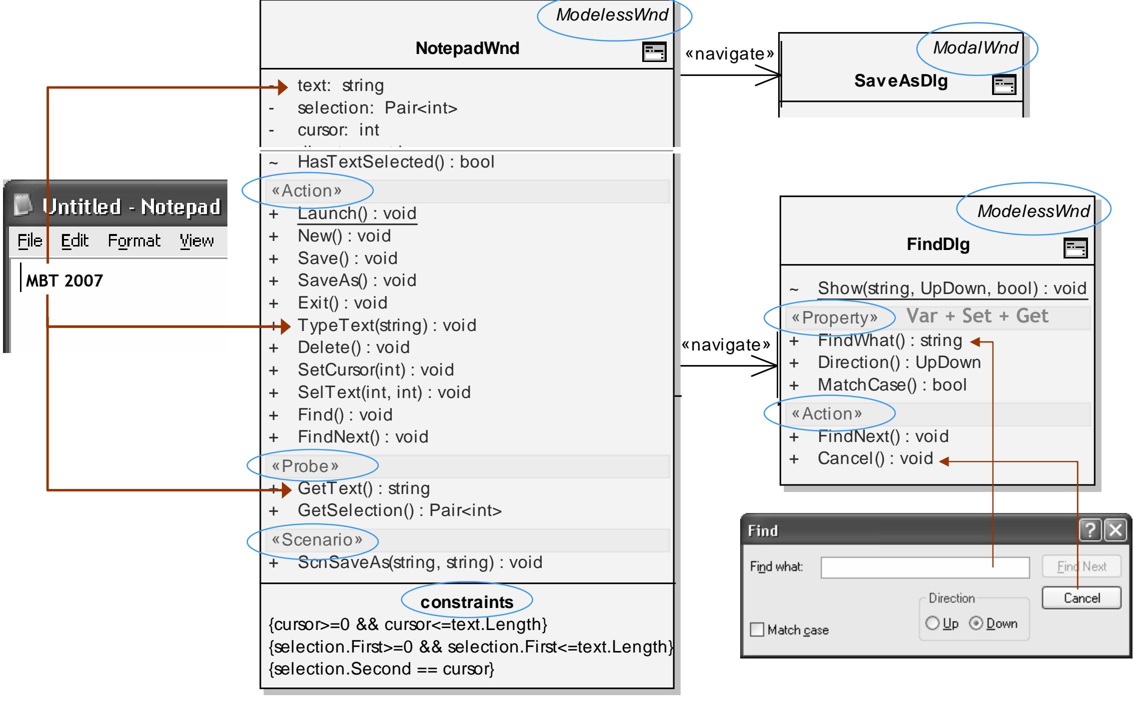
Scenario based testing

State based testing

Modelling GUI structure with UML

- Classes – Use to model top-level windows / UI components
  - «ModalWnd» - When open, the other windows of the application are disabled
  - «ModelessWnd» - When open, it's still possible to interact with other windows
- Attributes - Used to model internal window state / content
- Methods – Used mainly to model atomic and composite user actions
  - Spec#✓ «Action» - Describes the effect of an atomic user action on the GUI state (e.g., press a button, enter text)
  - Spec#✓ «Probe» - Models the observation of some GUI state from the user eyes (e.g., see / check the text in a textbox)
  - «Property» - Models a control with a value that can be set & read by the user (shorthand for attribute + set (Action) and get (Probe) methods)
  - Spec#✓ «Scenario» - Describes how a user should interact with the GUI to achieve a goal by a composition of lower level scenarios and atomic actions
- Associations – Used to model navigation
  - «navigate» - Models navigation among windows modelled as classes.

Modelling GUI structure with UML

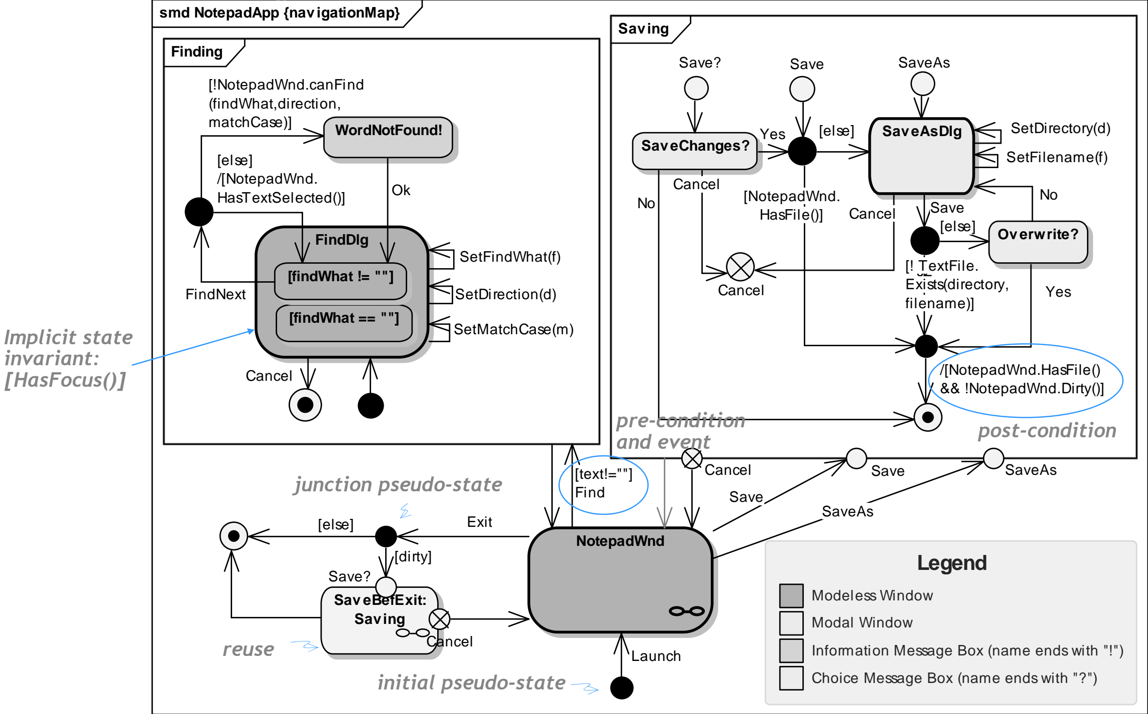


Modelling GUI behaviour with UML

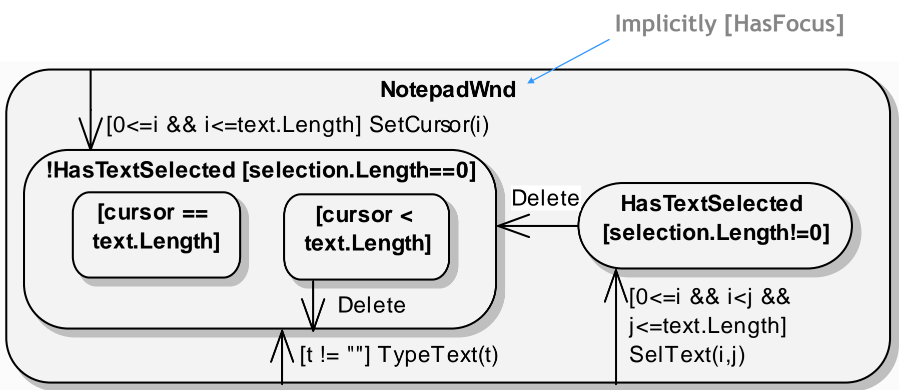
- UML 2.0 state machine diagrams
  - Based on statecharts, a recognized means to model the reactive behaviour of UIs
- Protocol (instead of behavioural) state machines
  - States may be formalized by conditions on the state variables (*state invariants*)
  - Transition are triggered by method invocations (representing user actions here)
  - Transitions may have pre/post-conditions on state vars and method parameters

  Advantages:
  - Abstraction - effects are specified implicitly via post-conditions, instead of explicitly via system actions
  - Separation of concerns - between the visual model (pre/post-conditions) and the refinements to introduce in the textual model (executable method bodies)
  - Convenience - to express black-box test goals (states and transitions as equivalence classes)
- Hierarchical
  - Top level state machine - Navigation map of the application
  - Next level – Internal window behaviour

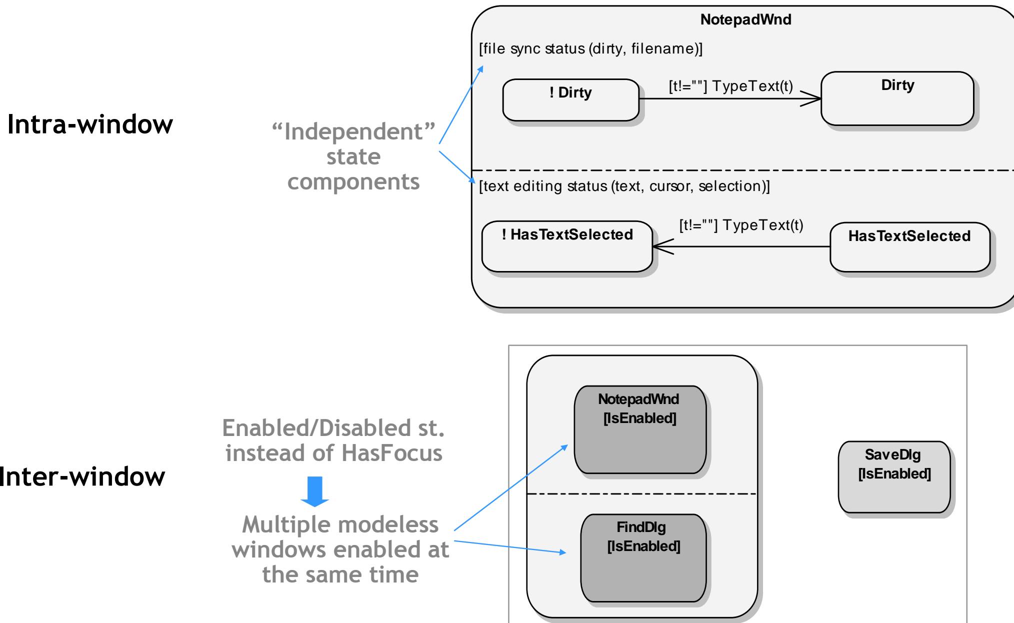
Modelling GUI behaviour: Inter-window



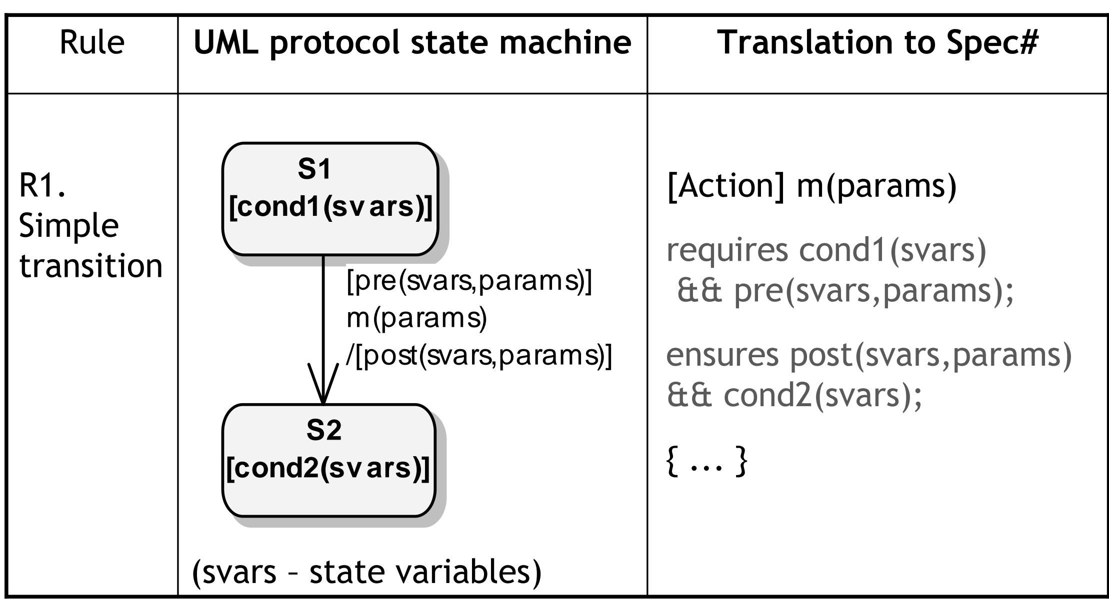
Modelling GUI behaviour: Intra-window



Modelling GUI behaviour: Orthogonal (concurrent) regions



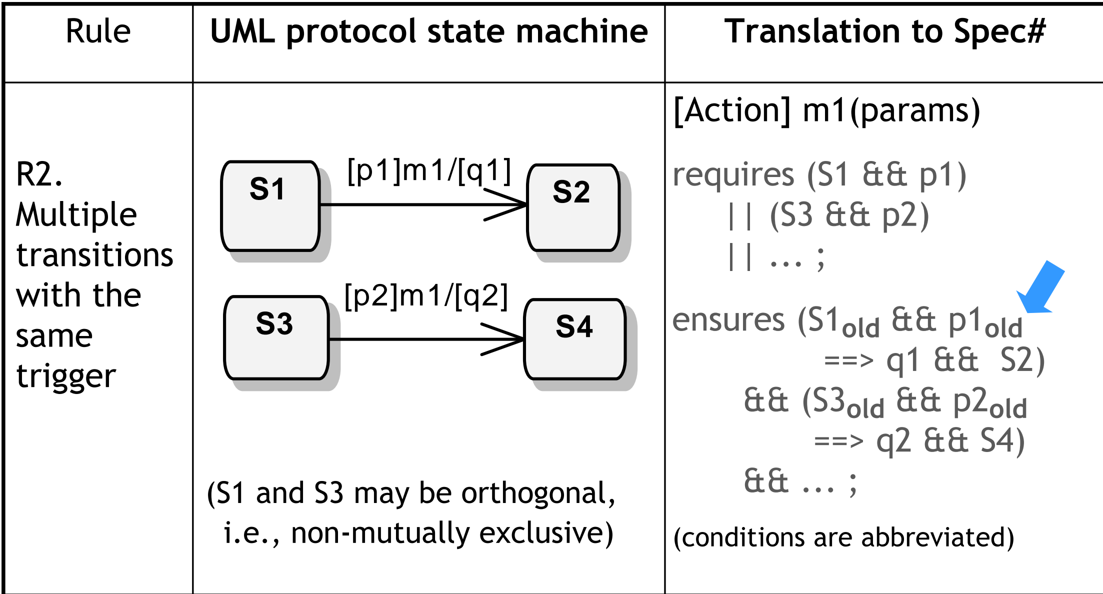
Protocol state machine translation (1)



Rule	UML protocol state machine	Translation to Spec#

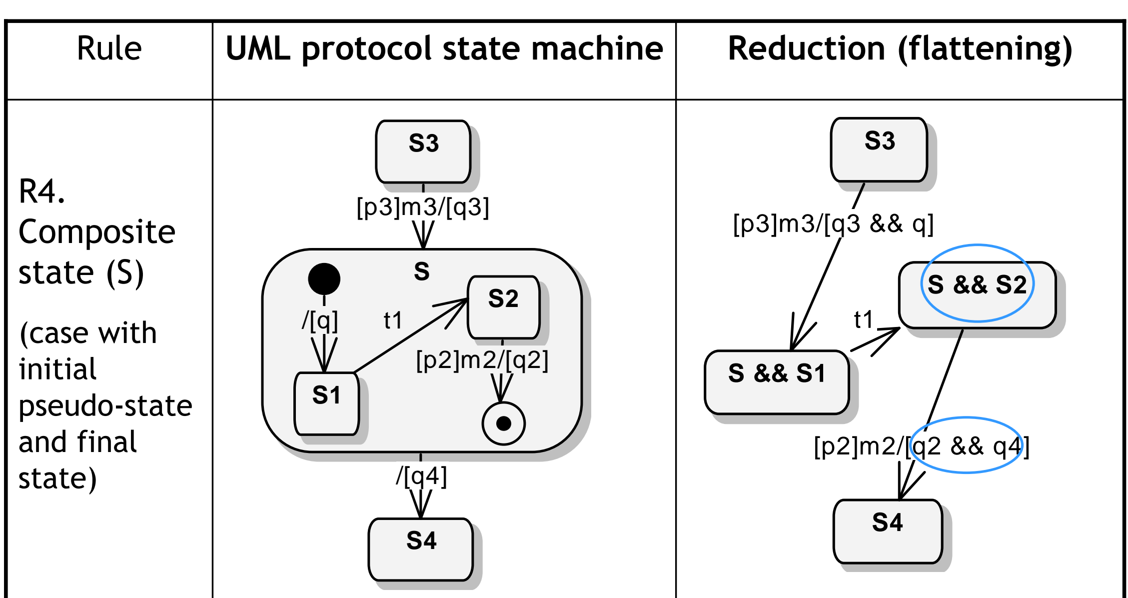
R1. Simple transition	S1 [cond1(svars)] → [pre(svars,params)] m(params) /[post(svars,params)] → S2 [cond2(svars)]
	(svars – state variables)
		[Action] m(params)
		requires cond1(svars) && pre(svars,params);
		ensures post(svars,params) && cond2(svars);
		{ … }

Protocol state machine translation (2)



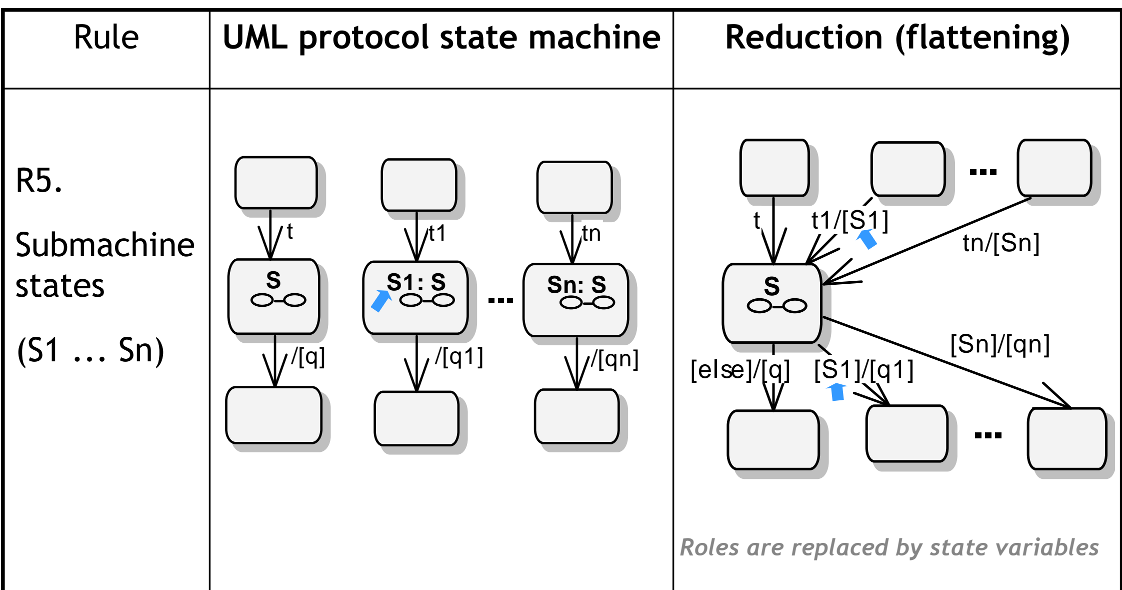
Rule	UML protocol state machine	Translation to Spec#
R2. Multiple transitions with the same trigger	S1 [p1]m1/[q1] S2 S3 [p2]m1/[q2] S4 (S1 and S3 may be orthogonal, i.e., non-mutually exclusive)	[Action] m1(params) requires (S1 && p1) || (S3 && p2) || … ; ensures ($S1_{old}$ && $p1_{old}$ ==> q1 && S2) && ($S3_{old}$ && $p2_{old}$ ==> q2 && S4) && … ; (conditions are abbreviated)

Protocol state machine translation (3)

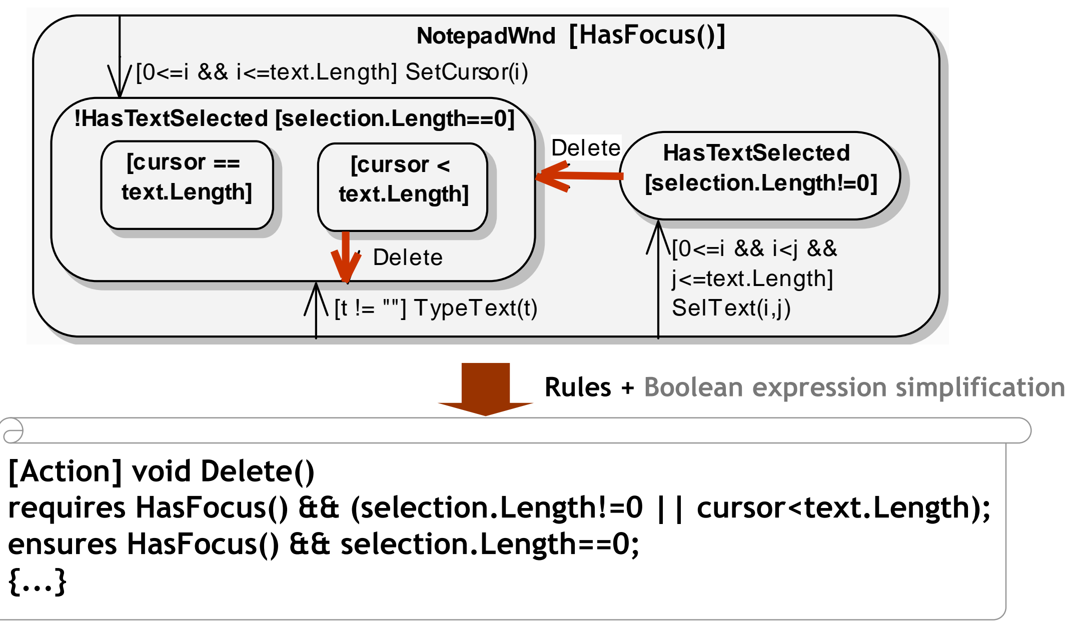


Rule	UML protocol state machine	Reduction (flattening)
R4. Composite state (S) (case with initial pseudo-state and final state)	S3 [p3]m3/[q3] S /[q] S1 t1 S2 [p2]m2/[q2] /[q4] S4	S3 [p3]m3/[q3 && q] S && S1 t1 S && S2 [p2]m2/[q2 && q4] S4

Protocol state machine translation (4)

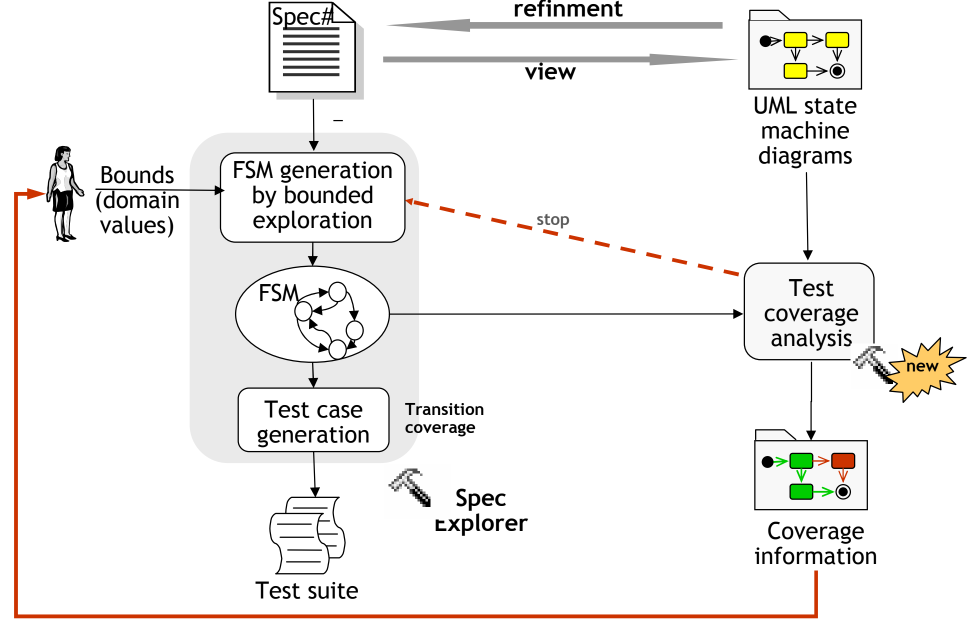
Rule	UML protocol state machine	Reduction (flattening)
R5. Submachine states (S1 … Sn)		*Roles are replaced by state variables*

Protocol state machine translation: Example

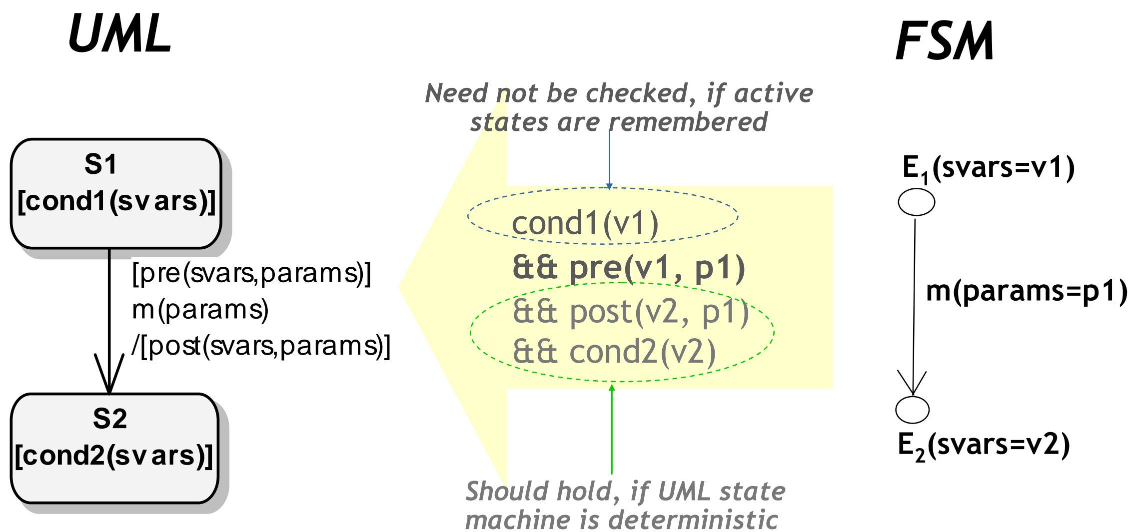



```
[Action] void Delete()
requires HasFocus() && (selection.Length!=0 || cursor<text.Length);
ensures HasFocus() && selection.Length==0;
{...}
```

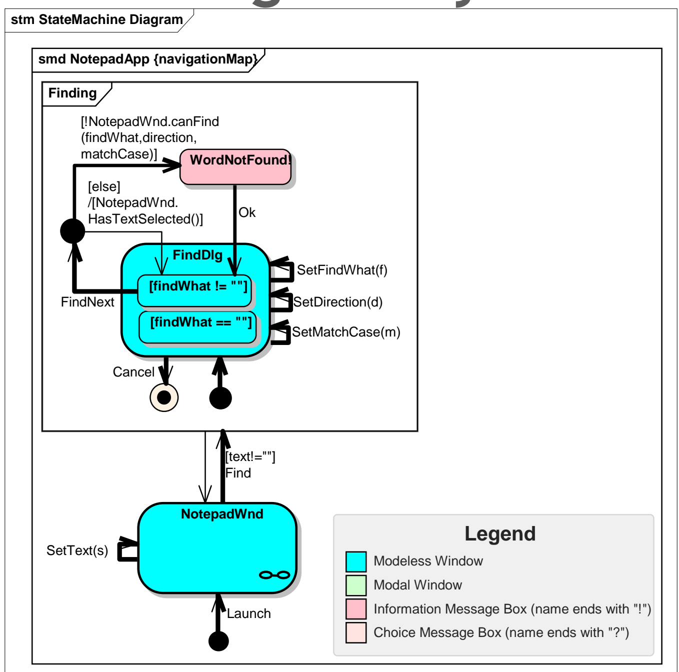
State based test case generation and coverage analysis



State based test case generation and coverage analysis

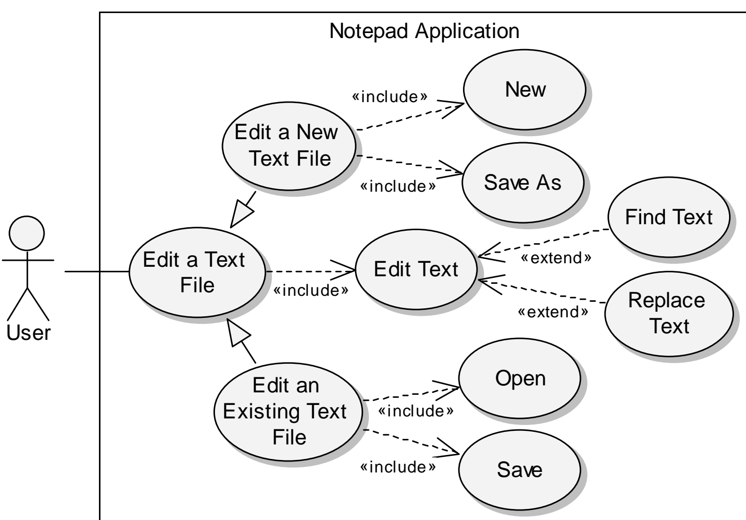


State based test case generation and coverage analysis

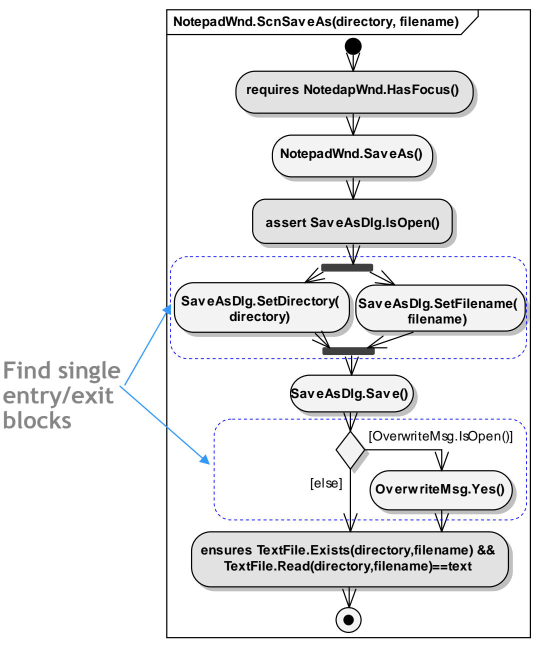


Launch()
NotepadWnd.SetText(s)
NotepadWnd.Find()
FindDlg.SetMatchCase(m)
FindDlg.SetDirection(d)
FindDlg.SetFindWhat(f)
FindDlg.FindNext()
WordNotFound.Ok()
FindDlg.Cancel()

Modelling GUI usage for Scenario BT: Use case diagram example



Modelling GUI usage for Scenario BT: Activity diagram & translation example




```
[Action(Kind=ActionAttributeKind.Scenario)]
void ScnSaveAs(string directory,string filename)
 requires NotepadWnd.HasFocus();
 ensures TextFile.Exists(directory,filename) &&
         TextFile.Read(directory,filename)== text;
// "text" is the state variable that holds the content
// in the edit area
{
  NotepadWnd.SaveAs();
  assert SaveAsDlg.IsOpen();
  parallel {
    SaveAsDlg.SetDirectory(directory);
    SaveAsDlg.SetFilename(filename);
  }
  SaveAsDlg.Save();
  if (OverwriteMsg.IsOpen()) {
      OverwriteMsg().Yes();
  }
}
```

Parameterized Unit Tests for UnitMeister?

Conclusions and future work

- Current status:
  - Small experiments conducted
  - Tools under development
- Future work
  - Round-trip engineering capabilities
  - Explore other visual behaviour modelling techniques
    - Use UML behavioural state machines, in order to completely hide the Spec# model at the price of a more detailed visual model (with procedural actions and decisions on transitions)
    - Produce less coupled models (with the exchange of signals between concurrent state machines) at the price of a more complex execution model
  - Reduce further the modelling effort for already existing GUIs by extracting a partial GUI model by a reverse engineering process
  - Validate the approach in industrial environments (AMBER iTest project)

Thank you!

PORTO World Heritage