Hierarchical GUI Test Case Generation Using Automated Planning[1]

Christian Halaschek-Wiener

1. Memon, A.M., Pollack, M.E., and Soffa, M.L. "Hierarchical GUI test case generation using automated planning", 144-155, IEEE Transactions on Software Engineering, Volume: 27, Issue: 2, Feb 2001

Motivation

- GUI testing inherently hard
  - Many items to test
    - Test underlying software
    - Must conform with GUI specs
  - Huge space of interactions with GUI
    - Large coverage area for test cases

Motivation

- Test Case Generation
  - Manual creation time consuming
- Present technique to automatically generate test cases

Approach Overview

- GUI consists of components that generate events
  - Lead to other GUIs or actions
- Given a GUI, its actions, and a goal
  - AI Planner generates paths through the system that achieve the goal
    - Each path is a test case

AI Planning Overview

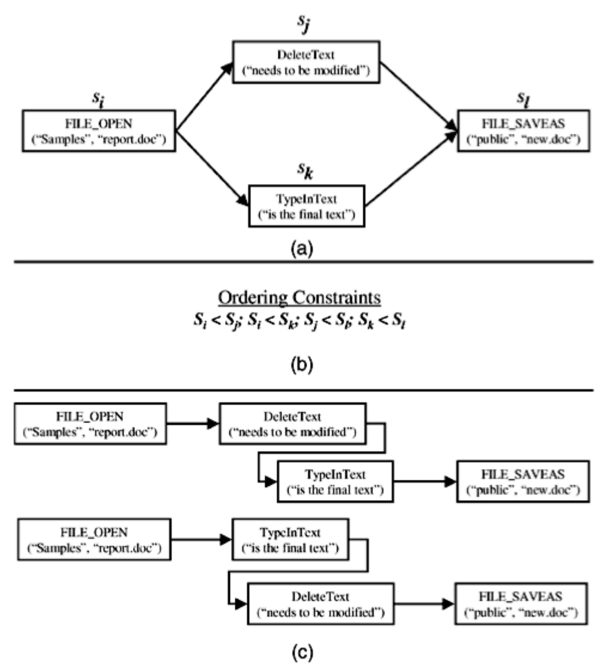
- Input
  - Set of objects
  - Set of operators
  - Initial state
  - Goal state
- Objects are GUI components/events
- Operators are the composition/actions of using them

AI Planning Overview

- Output
  - Set of plan steps (operators)
  - Ordering constraints on steps
  - Causal links
    - Structure of plan
  - Set of binding constraints on variables of operators
- Ordering constraints
  - State S_i occurs before S_k
- Causal links
  - Two states with a post-condition of S_i that is a pre-condition of S_k

AI Planning Overview

- Results in partially-ordered plans
- Can use/add ordering constraints to create total-order plans



Hierarchical Task Network (HTN) Planning

- Actions are modeled at different levels of abstraction
  - Operators at level *n*
    - Use one or more methods at level *n-1*
- GUI planning can be modeled as HTN planning problem
  - Provides greater planning efficiency
  - Modifications to GUI only require modifications to sub-plans

Steps

Phase	Step	Test Designer	PATHS
Setup	1		Derive Hierarchical GUI Operators
	2	Define Preconditions and Effects of Operators	
Plan Generation	3	Identify a Task $\mathcal{T}$	
	4		Generate Test Cases for $\mathcal{T}$

GUI Operators

- First construct operator set
  - Made up of GUI events
  - Note, one operator for each GUI event is inefficient
    - Exploit structural properties of GUI to create operators at different levels
  - Partition events into classes based on structural properties

GUI Events

- Different classes of events
  - Menu-open
    - Expand GUI events
      - e.g. File, Edit…
  - Unrestricted-focus
    - Open GUI window, not restricting focus
      - e.g. PPT basic shapes
  - System-interaction
    - Underlying software function
  - Restricted-focus
    - Preferences dialog

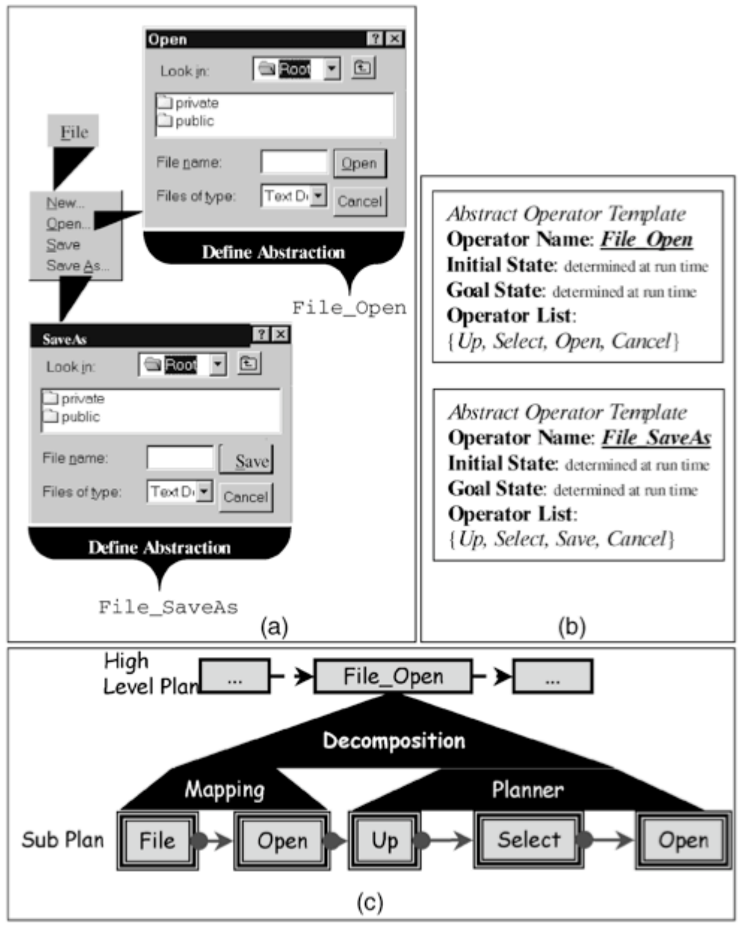
Operator Classes

- Two classes of planning operators
  - System-interaction
    - All sequences of zero or more menu-open and unrestricted events followed by a system interaction event
    - Example
      - Edit (menu-open) select, Paste (sys-int)
        - Results in operator EDIT_PASTE = <Edit, Paste>
    - Use an operator event mapping
      - Reduces number of operators made available to the planner

Operator Classes

- Abstract Operator
  - Restricted-focus events
  - Encapsulate events by creating new planning problem
    - Layer of abstraction
  - Essentially is sub-planning problem
    - Prefix - sequence of menu-open and unrestricted-focus that lead to this event
    - Suffix - the restricted-focus user interaction
      - Planner determines the suffix
    - Both are substituted back into the plan

Abstract Operator



Operator-Event Mappings

GUI Events = {File, Edit,
New, Open, Save, SaveAs,
Cut, Copy, Paste,
Open.Up, Open.Select, Open.Cancel, Open.Open,
SaveAs.Up, SaveAs.Select, SaveAs.Cancel, SaveAs.Save}.

(a)

Planning Operators = {
File_New, File_Open, File_Save, File_SaveAs,
Edit_Cut, Edit_Copy, Edit_Paste}.

(b)

Operator Name	Operator Type	GUI Events
FILE_NEW	Sys. Interaction	<File, New>
FILE_OPEN	Abstract	<File, Open>
FILE_SAVE	Sys. Interaction	<File, Save>
FILE_SAVEAS	Abstract	<File, SaveAs>
EDIT_CUT	Sys. Interaction	<Edit, Cut>
EDIT_COPY	Sys. Interaction	<Edit, Copy>
EDIT_PASTE	Sys. Interaction	<Edit, Paste>

Abstraction Advantages

- Wordpad
  - 10:1 reduction ration in operator number
  - 325 GUI events
  - Reduced to 32 system interaction and abstract operators

Preconditions and Effects

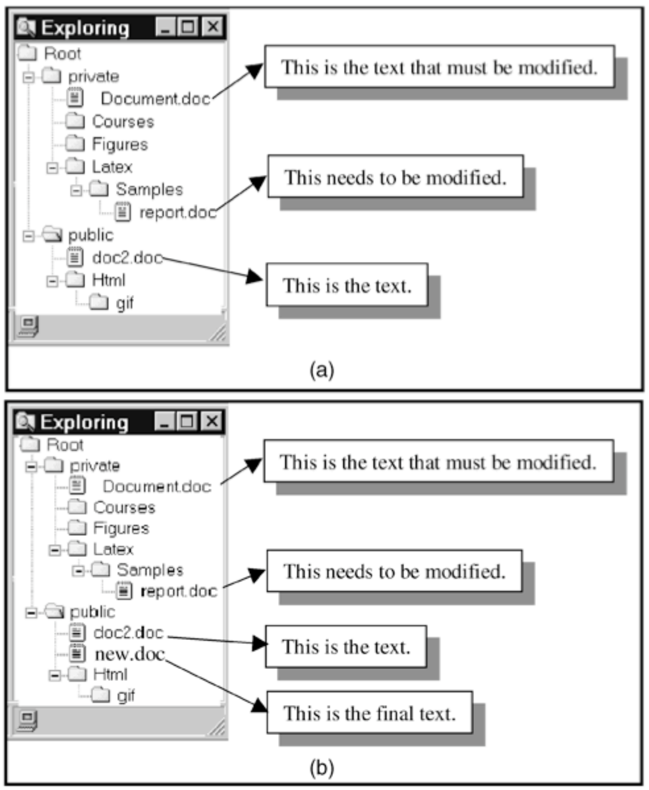
- Test designer must next specify the preconditions and effects for each operator
- Definition language provided by the planner

```
Operator :: EDIT_CUT
Preconditions:
        EXISTS Obj in Objects
                Selected(Obj).

Effects:
        FORALL Obj in Objects
                Selected(Obj) ⇒
                        ADD inClipboard(Obj)
                        DEL onScreen(Obj)
                        DEL Selected(Obj).
```

Identify Planner Task

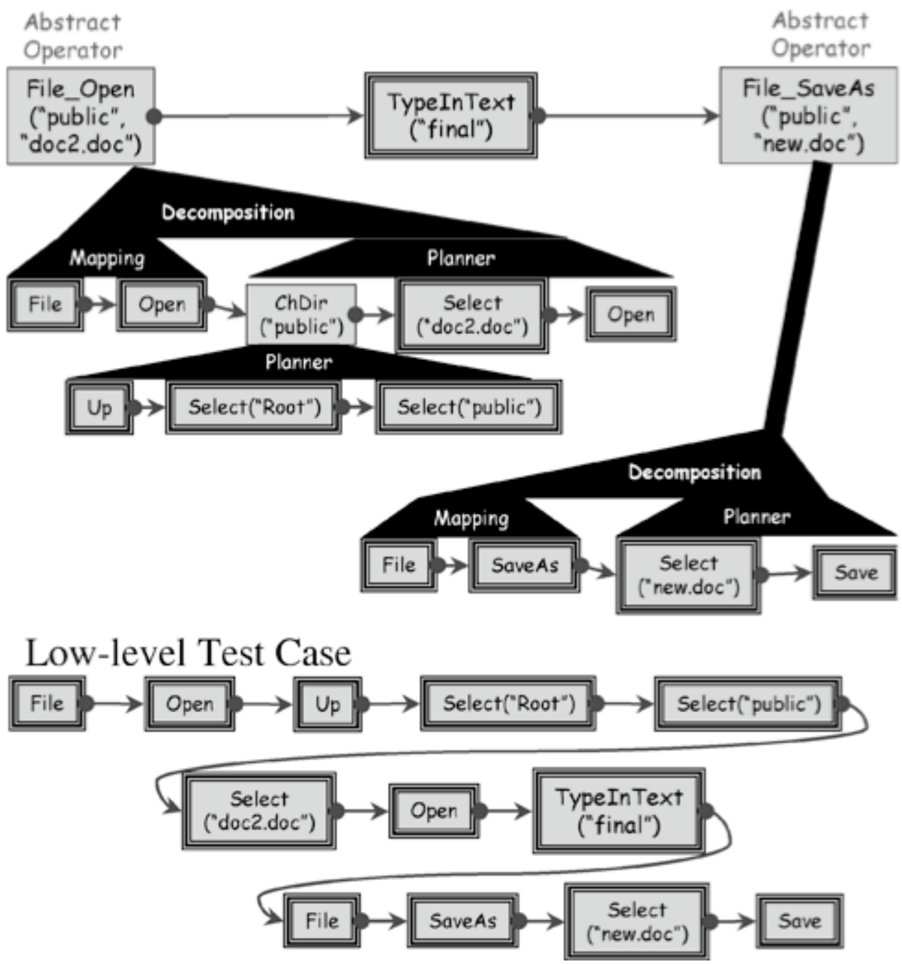
- Consists of initial and goal states




```
Initial State:
isCurrent(root)
contains(root private)
contains(private Figures)
contains(private Latex)
contains(Latex Samples)
contains(private Courses)
contains(private Thesis)
contains(root public)
contains(public html)
contains(html gif)
containsfile(gif doc2.doc)
containsfile(private
            Document.doc)
containsfile(Samples
report.doc)
currentFont(Times Normal
            12pt)
in(doc2.doc This)
in(doc2.doc is)
in(doc2.doc the)
in(doc2.doc text.)
isText(This)
isText(is)
isText(the)
isText(text)
after(This is)
after(is the)
after(the text.)
font(This Times Normal 12pt)
font(is Times Normal 12pt)
font(the Times Normal 12pt)
font(text. Times Normal
            12pt)
.............
Similar descriptions for
Document.doc and report.doc

Goal State:
containsfile(public new.doc)
in(new.doc This)
in(new.doc is)
in(new.doc the)
in(new.doc final)
in(new.doc text.)
after(This is)
after(is the)
after(the final)
after(final text.)
font(This Times Normal 12pt)
font(is Times Normal 12pt)
font(the Times Normal 12pt)
font(final Times Normal
            12pt)
font(text. Times Normal
            12pt)
.................
```

Big Picture



Abstract Operator
File_Open ("public", "doc2.doc")
TypeInText ("final")
Abstract Operator
File_SaveAs ("public", "new.doc")
Decomposition
Mapping
File
Open
Planner
ChDir ("public")
Select ("doc2.doc")
Open
Planner
Up
Select("Root")
Select("public")
Decomposition
Mapping
Planner
File
SaveAs
Select ("new.doc")
Save
Low-level Test Case
File
Open
Up
Select("Root")
Select("public")
Select ("doc2.doc")
Open
TypeInText ("final")
File
SaveAs
Select ("new.doc")
Save

Resulting Plans

Plan No.	Plan Step	Plan Action
1	1	FILE-OPEN("private", "Document.doc")
	2	DELETE-TEXT("that")
	2	DELETE-TEXT("must")
	2	DELETE-TEXT("be")
	2	DELETE-TEXT("modified")
	2	TYPE-IN-TEXT("final", Times, Italics, 12pt)
	3	FILE-SAVEAS("public", "new.doc")
2	1	FILE-OPEN("public", "doc2.doc")
	2	TYPE-IN-TEXT("is", Times, Italics, 12pt)
	2	TYPE-IN-TEXT("the", Times, Italics, 12pt)
	2	DELETE-TEXT("needs")
	2	DELETE-TEXT("to")
	2	DELETE-TEXT("be")
	2	DELETE-TEXT("modified")
	2	TYPE-IN-TEXT("final", Times, Italics, 12pt)
	2	TYPE-IN-TEXT("text", Times, Italics, 12pt)
	3	FILE-SAVEAS("public", "new.doc")
3	1	FILE-OPEN("public", "doc2.doc")
	2	TYPE-IN-TEXT("is", Times, Italics, 12pt)
	2	TYPE-IN-TEXT("the", Times, Italics, 12pt)
	2	DELETE-TEXT("to")
	2	DELETE-TEXT("be")
	2	DELETE-TEXT("modified")
	2	TYPE-IN-TEXT("final", Times, Italics, 12pt)
	2	TYPE-IN-TEXT("text", Times, Italics, 12pt)
	2	SELECT-TEXT("needs")
	3	EDIT-CUT("needs")
	4	FILE-SAVEAS("public", "new.doc")
4	1	FILE-NEW("public", "new.doc")
	2	TYPE-IN-TEXT("This", Times, Italics, 12pt)
	2	TYPE-IN-TEXT("is", Times, Italics, 12pt)
	2	TYPE-IN-TEXT("the", Times, Italics, 12pt)
	2	TYPE-IN-TEXT("final", Times, Italics, 12pt)
	2	TYPE-IN-TEXT("text", Times, Italics, 12pt)
	3	FILE-SAVEAS("public", "new.doc")

Results

- Time to generate plans

Task No.	Plan Time (sec)	Sub Plan Time	Total Time (sec)
1	0.40	0.04	0.44
2	3.16	0.00	3.16
3	3.17	0.00	3.17
4	3.20	0.01	3.21
5	3.38	0.01	3.39
6	3.44	0.02	3.46
7	4.09	0.04	4.13
8	8.88	0.02	8.90
9	40.47	0.04	40.51

Performance Results

- Single level vs. Hierarchical

	Single level		Hierarchical	
Task No.	Plan Length	Time (sec.)	Plan Length	Time (sec.)
1	18	8.93	3	0.11
2	20	47.62	4	0.18
3	24	189.87	5	0.14
4	26	3312.72	6	7.18
5	-	-	3	0.1
6	-	-	4	13.01

- Due to reduction in computation space because of abstraction - 10:1 reduction

Results

- Also note that they implemented an automated test execution system
  - Generate mouse/keyboard events

Limitations

- Test cases generated depend on tasks defined by user
  - Poor tasks…poor coverage
    - Developing coverage measures
- Depend heavily on hierarchical structure of GUI
  - If poorly structured, no abstract operators
- Should be used in conjunction with other generators

Critique

- Frequency that operator abstraction can actually be used
- User involvement
  - State commonly used operators could be maintained in libraries

Related Work

- Record/Playback tools
  - Sessions are played back to recreate GUI steps
- FSM
  - Used for test case generation
  - Once built, generation process automatic
  - Scaling problems
- Genetic Algorithms
  - Expert generates initial GUI events
  - GA extends and modifies them
  - Assumption - Expert takes short path, where novice will take longer ones

Questions & Comments

1. Memon, A.M., Pollack, M.E., and Soffa, M.L. “Hierarchical GUI test case generation using automated planning”, 144-155, IEEE Transactions on Software Engineering, Volume: 27, Issue: 2, Feb 2001