



Exago Exercises

Anikó Nagyné Víg and Atilla Erdődi

Stockholm, 13 Nov 2009



The “classic” elevator example

Modifications:

- Events are captured to file for analysis
- Unique id is generated for each event



Deciding the model

We abstract out the buttons, the command queue and only care about *what* happened but not *why*.

Copyright 2009



Exercise 1: Define the model in Exago

0. Generate some log files

- Start the elevator example with tracing:
 - `util:start_trace(1,3,3).`
- Call an elevator to floor 2
- Send elevator 2 to floor 2
- Send elevator 3 to floor 3

Copyright 2009



Log file format

1. Define the log file in the tool

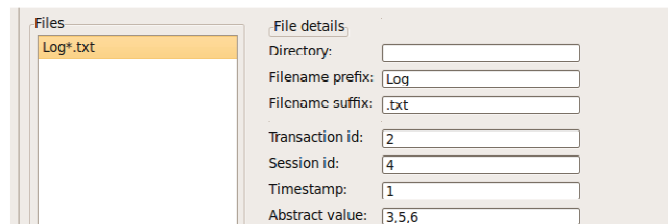
- Only one file:
 - Log.txt (or as set in the app file)
- Format:
 - Timestamp, EventId, Event, ElevatorId, [Parameters]

Copyright 2009



Log file format

- ElevatorId → Session Id
 - We don't have *real* sessions in this application
- EventId → Transaction Id



The screenshot shows a file configuration interface. On the left, a 'Files' list contains 'Log*.txt'. On the right, a 'File details' section contains the following fields:

Directory:	<input type="text"/>
Filename prefix:	<input type="text" value="Log"/>
Filename suffix:	<input type="text" value=".txt"/>
Transaction id:	<input type="text" value="2"/>
Session id:	<input type="text" value="4"/>
Timestamp:	<input type="text" value="1"/>
Abstract value:	<input type="text" value="3,5,6"/>

Copyright 2009



Transaction abstraction

2. Define the abstraction functions

- Simple case: one event per transaction
- Trivial transaction abstraction function

```
1 fun(_TrId, [{Timestamp, SessionId, AbstrVal, File} | _]) ->
2   {Timestamp, AbstrVal}
3 end.
```

Copyright 2009



Transaction filtering

- Filtering out the events
we don't take into account in our model

```
1 fun({_Timestamp, AbstrVal}) ->
2   case AbstrVal of
3     {"reset", "closed", _Floor} -> true;
4     {"open", none, none} -> true;
5     {"close", none, none} -> true;
6     {"approaching", _Floor, none} -> true;
7     {"stopped at", _Floor, none} -> true;
8     _ -> false
9   end
10 end.
```

Copyright 2009



Session abstraction

- Creating the abstract commands

```
1 fun(Trs) ->
2   lists:map(fun({Timestamp, AbstrVal}) ->
3     {Timestamp,
4       case AbstrVal of
5         {"reset", "closed", Floor} ->
6           list_to_atom("reset to_" ++ Floor);
7         {"open", none, none} ->
8           open;
9         {"close", none, none} ->
10          close;
11         {"approaching", Floor, none} ->
12           list_to_atom("approaching_" ++ Floor);
13         {"stopped at", Floor, none} ->
14           list_to_atom("stopped_at_" ++ Floor);
15         other ->
16           _other
17       end}
18   end, Trs)
19 end.
```

Copyright 2009



State machine

3. Define the state machine

- 10 states for 3 elevators and 3 floors

The interface displays the following data:

States	
Nr.	State name
0	init
1	Closed at floor 1
2	Open at floor 1
3	Approaching floor 1
4	Closed at floor 2
5	Open at floor 2
6	Approaching floor 2
7	Closed at floor 3
8	Open at floor 3
9	Approaching floor 3

Transitions				
From nr	To nr	Command	Timeout	
0	1	reset to 1		
0	4	reset_to_2		
0	7	reset_to_3		
1	2	open		
2	1	close		
3	1	stopped_at_1		
6	4	stopped_at_2		
4	5	open		
5	4	close		
4	3	approaching_1		
4	9	approaching_3		

Options

Terminal states:

Good terminal states:

Initial state:

Copyright 2009



Interpreting the results

4. Run the tool and analyse the results

Take a look at one of the failed sessions:

```
{"2009-11-11 14:22:22:0089135","reset_to_1"}  
{"2009-11-11 14:22:27:0713588","approaching_1"}  
{"2009-11-11 14:22:29:0345272","approaching_2"}  
{"2009-11-11 14:22:29:0956680","stopped_at_3"}  
{"2009-11-11 14:22:29:0957266","open"}  
{"2009-11-11 14:22:30:0960854","close"}  
{"2009-11-11 14:22:31:0985493","approaching_3"}  
{"2009-11-11 14:22:33:0621364","approaching_2"}  
...
```

Copyright 2009



Interpreting the results

The log files claims the elevator tries to approach floor 1...

```
{"2009-11-11 14:22:22:0089135","reset_to_1"}  
{"2009-11-11 14:22:27:0713588","approaching_1"}  
{"2009-11-11 14:22:29:0345272","approaching_2"}  
{"2009-11-11 14:22:29:0956680","stopped_at_3"}  
{"2009-11-11 14:22:29:0957266","open"}  
{"2009-11-11 14:22:30:0960854","close"}  
{"2009-11-11 14:22:31:0985493","approaching_3"}  
{"2009-11-11 14:22:33:0621364","approaching_2"}  
{"2009-11-11 14:22:34:0232767","stopped_at_1"}  
{"2009-11-11 14:22:34:0233367","open"}  
{"2009-11-11 14:22:35:0237494","close"}  
{"2009-11-11 14:22:36:0261430","approaching_1"}  
{"2009-11-11 14:22:37:0892879","approaching_2"}  
{"2009-11-11 14:22:38:0504843","stopped_at_3"} ...
```

Copyright 2009



Interpreting the results

... but we are already on floor 1!

```
{"2009-11-11 14:22:22:0089135","reset_to_1"}  
{"2009-11-11 14:22:27:0713588","approaching_1"}  
{"2009-11-11 14:22:29:0345272","approaching_2"}  
{"2009-11-11 14:22:29:0956680","stopped_at_3"}  
{"2009-11-11 14:22:29:0957266","open"}  
{"2009-11-11 14:22:30:0960854","close"}  
{"2009-11-11 14:22:31:0985493","approaching_3"}  
{"2009-11-11 14:22:33:0621364","approaching_2"}  
{"2009-11-11 14:22:34:0232767","stopped_at_1"}  
{"2009-11-11 14:22:34:0233367","open"}  
{"2009-11-11 14:22:35:0237494","close"}  
{"2009-11-11 14:22:36:0261430","approaching_1"}  
{"2009-11-11 14:22:37:0892879","approaching_2"}  
{"2009-11-11 14:22:38:0504843","stopped_at_3"} ...
```

Copyright 2009



Finding the cause

The abstract commands do not show the the cause of the error in this case, neither the experienced “symptoms”, but clearly indicates that the elevator does not match even with this basic state machine.

Copyright 2009



Finding the cause

However, we can see there is something wrong when the elevator leaves the floor.

Copyright 2009



Examining the code

The bug is in the elevator.erl:

```
moving({approaching, Floor}, {ENo, NewFloor}) -> %% Oops...
  sys_event:approaching(ENo, NewFloor),
  case scheduler:approaching(ENo, NewFloor) of
  {ok, stop} ->
    sys_event:stopping(ENo),
    {next_state, stopping, {ENo, Floor}};
  {ok, continue} ->
    {next_state, moving, {ENo, Floor}};
  _Other ->
    sys_event:stopping(ENo),
    {next_state, stopping, {ENo, Floor}}
  end;
```

Copyright 2009



Exercise 2: add time constraints

The elevator door should remain opened
for at least 3 seconds

<=>

At least 3 second should pass between
state „open” and state „closed”

Copyright 2009

Erlang

Exercise 2: solution

The screenshot shows a state machine editor with the following components:

- States Table:**

Nr.	State name
0	Init
1	Closed at floor 1
2	Open at floor 1
3	Approaching floor 1
4	Closed at floor 2
5	Open at floor 2
6	Approaching floor 2
7	Closed at floor 3
8	Open at floor 3
9	Approaching floor 3

- Transitions Table:**

From nr	To nr	Command	Timeout
0	1	reset_to_1	
0	4	reset_to_2	
0	7	reset_to_3	
1	2	open	
2	1	close	{gt,30}
3	1	stopped_at_1	
6	4	stopped_at_2	
4	5	open	
5	4	close	{gt,30}
4	3	approaching_1	
4	9	approaching_3	

- Options:**
 - Terminal states: 0,1,2,3,4,5,6,7,8,9
 - Good terminal states: 0,1,2,3,4,5,6,7,8,9
 - Initial state: 0

Copyright 2009

Erlang

Thank you for your attention!

Any questions?

Copyright 2009

Erlang