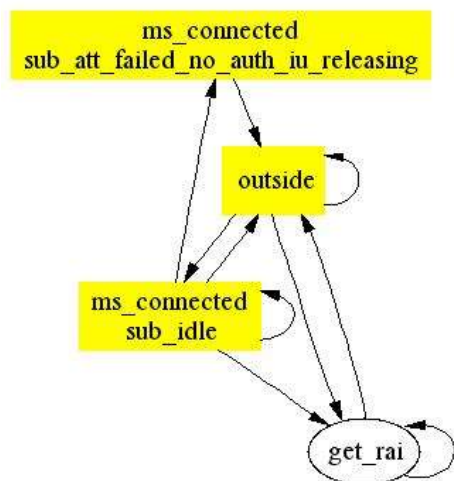




IT-universitetet
i Göteborg

CHALMERS | GÖTEBORGS UNIVERSITET

Reverse engineering Erlang software

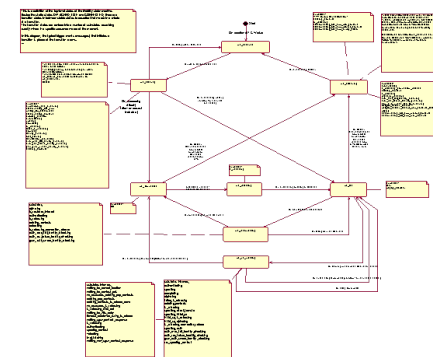


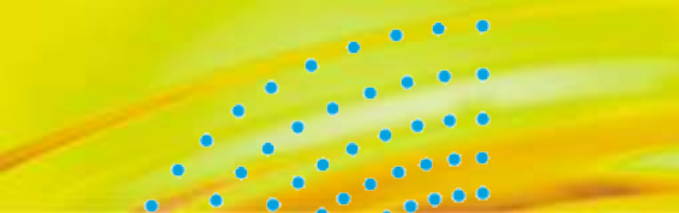
Thomas Arts

IT university in Göteborg

Cecilia Holmqvist

Ericsson AB





IT-universitetet
i Göteborg

CHALMERS | GÖTEBORGS UNIVERSITET

Design out of date...

Ericsson has developed nodes for the UMTS and GPRS network in Erlang

*Development in three countries over the last years. Now concentrated on
Lindholmen (Göteborg).*

*Designed in UML, implemented in Erlang. **The code has been
changed, not the design.***

A decorative graphic consisting of a cluster of blue dots of varying sizes, arranged in a pattern that suggests a network or data flow, located in the top left corner of the slide.

Design out of date...

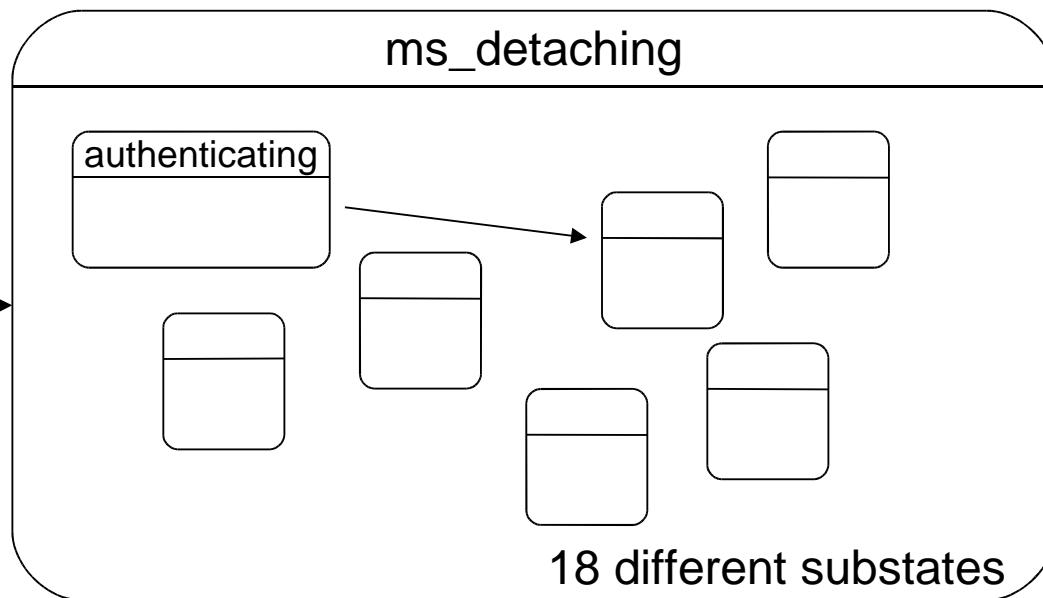
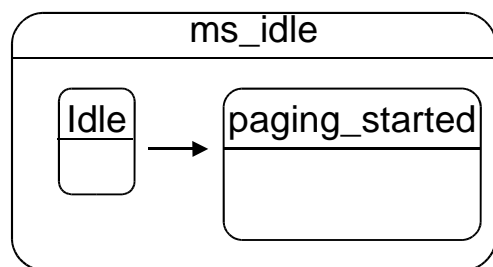
Can we extract the design from the code?

Useful for

- *better understanding of the system,*
- *re-implementation of the system,*
- *documentation purposes,*
- *differences indicate possible problems*

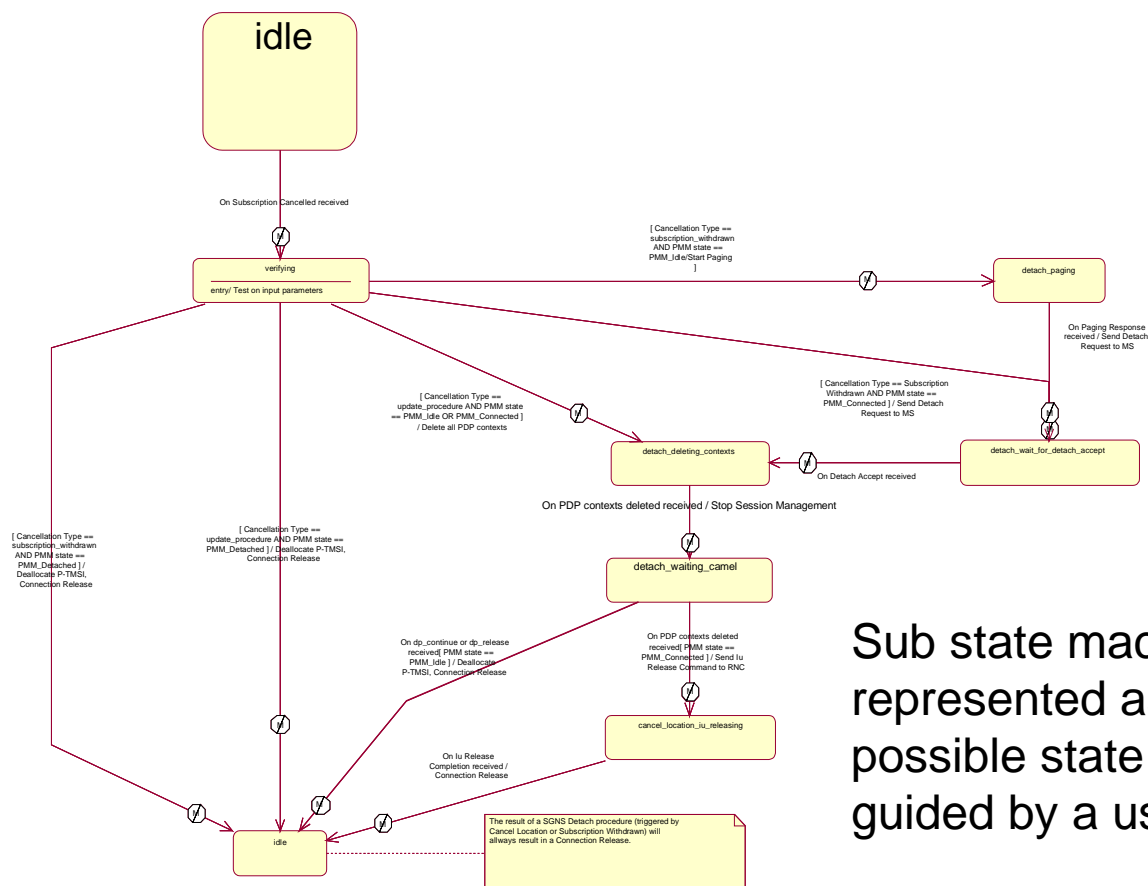
Mobility Management

Hierarchical State Machines



Mobility Management

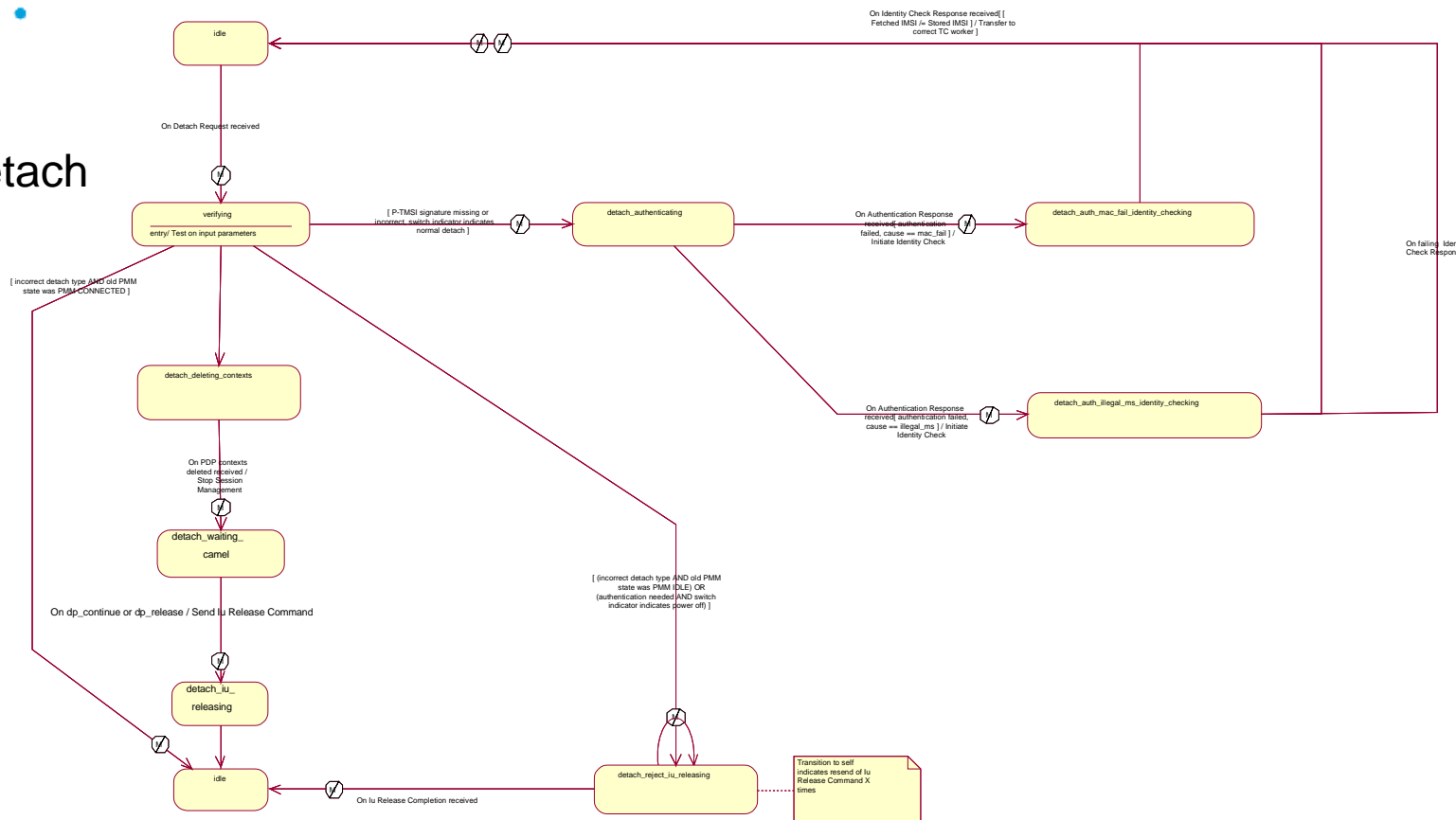
ms_detach

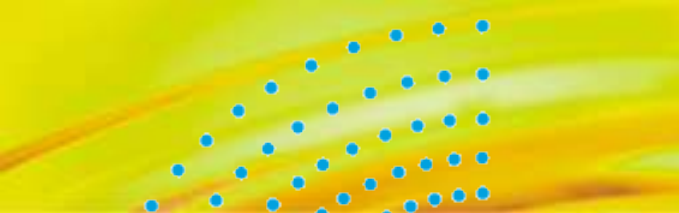


Sub state machines represented as a set of possible state transitions guided by a use-case

Mobility Management

ms_detach





IT-universitetet
i Göteborg

CHALMERS | GÖTEBORGS UNIVERSITET

Extract the design

Can we generate similar pictures, i.e., generate a state model for a certain use-case ?

Source code analysis has been studied before, e.g. Nyström 2001 and Mohagheghi et al 2003

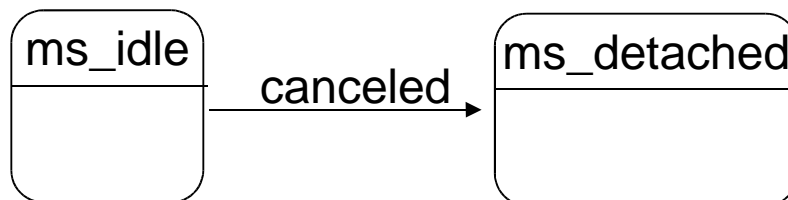
Runtime analysis has been studied in earlier work Arts and Fredlund 2002

Source code analysis

Example:

Given the source code of a generic finite state machine, one can extract a picture of the finite state machine

```
ms_idle( { canceled, ... } , Data) ->  
.....  
{ next_state, ms_detached, ... } ;
```





Source code analysis

Difficult to use here, because state machine implemented by many different modules, in a very non-standard way.

Events are implemented as function calls, e.g. `detach_request / 6`

Cascade of function calls in several modules follow such an event

*One of the function calls in the cascade can be `mmumoc:set_state / 2`
which registers the state*

Flow analysis almost impossible



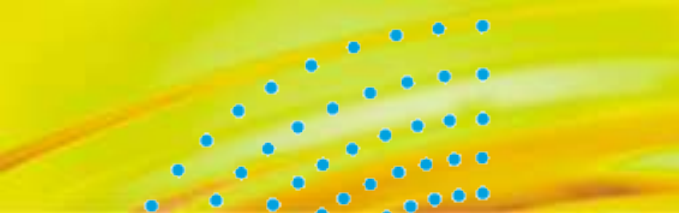
IT-universitetet
i Göteborg

CHALMERS | GÖTEBORGS UNIVERSITET

Source code analysis

*The diagrams one wants to obtain reflect a flow through the state machine
when dealing with a certain use-case*

*With pure symbolic analysis one obtains the complete state machine instead
of one specific for a scenario*



IT-universitetet
i Göteborg

CHALMERS | GÖTEBORGS UNIVERSITET

Runtime analysis

Idea: run the software on a special test case obtained from the use-case

Register all events and state changes that occur

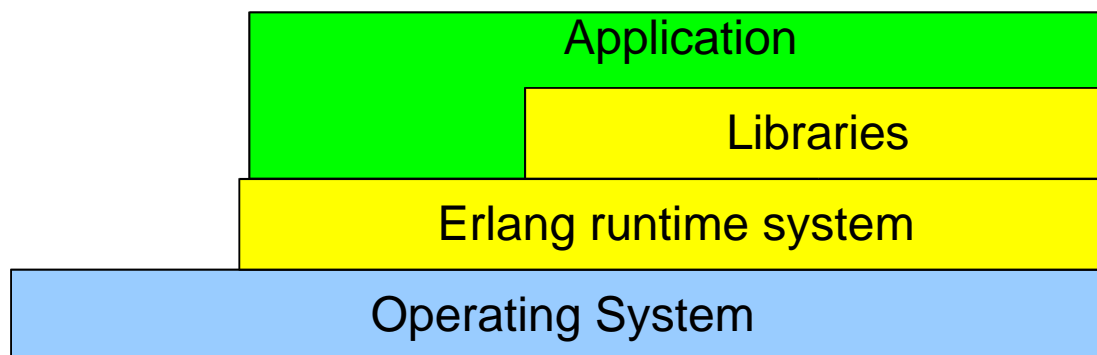
Test cases are already developed

Erlang has an advanced tracing possibility



Runtime analysis

The functions `erlang_trace/2` and `erlang_trace_pattern/2` can be used to send a message to a collection process every time a certain function is called.



Runtime analysis

Standard trace tool used to trace calls to MMU modules while executing test cases for a certain use-case

```
TRACE 2004-02-16 14:41:01,915166 <3730.1335.0>,call,mmumoc_c:modify_node_prop
Caller.....:noncos_c_nodeprop:modify_node_prop/3
Node.....:'e_Erlang_Global_pm1_1_2_1@octavo151.0'
Argument List...:[void,"SelectiveAuthenticationFrequency","10"]

TRACE 2004-02-16 14:41:01,915215 <3730.1335.0>,call,mmumoc_c:check_values
Caller.....:mmumoc_c:modify_node_prop/3
Node.....:'e_Erlang_Global_pm1_1_2_1@octavo151.0'
Argument List...:["SelectiveAuthenticationFrequency",10]

TRACE 2004-02-16 14:41:01,915252 <3730.1335.0>,return_from,mmumoc_c:check_values/2
Node.....:'e_Erlang_Global_pm1_1_2_1@octavo151.0'
Return Value...:{valid,10}

TRACE 2004-02-16 14:41:01,917425 <3730.1335.0>,return_from,mmumoc_c:modify_node_prop/3
Node.....:'e_Erlang_Global_pm1_1_2_1@octavo151.0'
Return Value...:{ok,"void"}
```



IT-universitetet
i Göteborg

CHALMERS | GÖTEBORGS UNIVERSITET

Trace data

Average 15,000 entries, file size 9MB

Erlang log-file format (binary)

Analyzing with emacs rather unpleasant

We are interested in the calls to `mmumoc:set_state/2`

Filter Trace data

Prepared for 1GB or larger log files, filter out the functions in which state is set.

```
read(FileName, Predicate) ->
  {ok, FileDescr} =
    file:open(FileName, [read, raw, binary]),
  Terms =
    unpack(FileDescr, Predicate),
  file:close(FileDescr),
  Terms.
```

Filter Trace data

```
unpack(FileDescr, Predicate) ->
  case file:read(FileDescr,5) of
    {ok, <<B1,B2,B3,Size:16>>} ->
      {ok, BTerm} =
        file:read(FileDescr, Size),
      Term =
        binary_to_term(BTerm),
      case Predicate(Term) of
        true ->
          [Term|unpack(FileDescr, Filter)];
        false ->
          unpack(FileDescr, Filter)
      end;
    eof ->
      []
  end.
```



Filter Trace data

Predicate example:

```
state_mmu() ->  
  fun({trace_ts,Pid,call,{mmumoc,set_state,[S,SS]},Caller,TS}) ->  
    true;  
    (_) ->  
      false  
  end.
```

Combining several predicates (or, and, not):

```
pred_or(F1,F2) ->  
  fun(T) ->  
    F1(T) or F2(T)  
  end.
```



Abstract trace data

Similar to filtering, we define abstraction functions that are applied to all entries in the trace.

For example:

```
{trace_ts, Pid, call, {mmumoc, set_state,  
  [S, SS]}, Caller, TS}
```

can be abstracted to

```
{state, [S, SS]}
```



IT-universitetet
i Göteborg

CHALMERS | GÖTEBORGS UNIVERSITET

Abstract trace data

Abstracting trace entries allows to map different entries to the same constant.

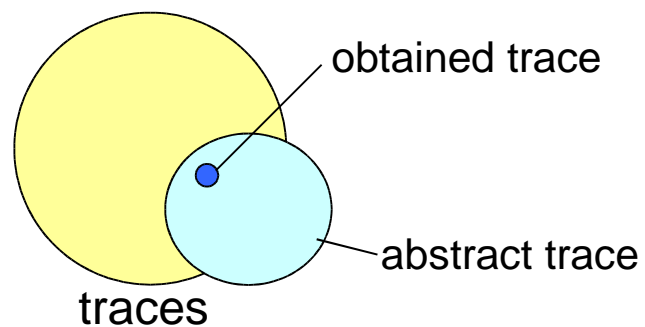
For example:

All states but `ms_connected` are abstracted to outside

This selects one state with all substates from a trace

Abstract trace data

More than one trace



Abstract trace data

```
a(event, X) ->
  {next_state, b, X}.
```

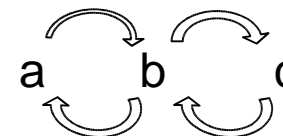
```
b(event, X) ->
  case X rem 2 of
    0 -> {next_state, a, X+1};
    1 -> {next_state, c, X+1}
  end.
```

```
c(event, X) ->
  {next_state, b, X}.
```

Trace:

```
=
a(event, 1)
b(event, 1)
c(event, 2)
b(event, 2)
a(event, 3)
b(event, 3)
c(event, 4)
b(event, 4)
a(event, 5)
b(event, 5)
```

....





Structuring trace data

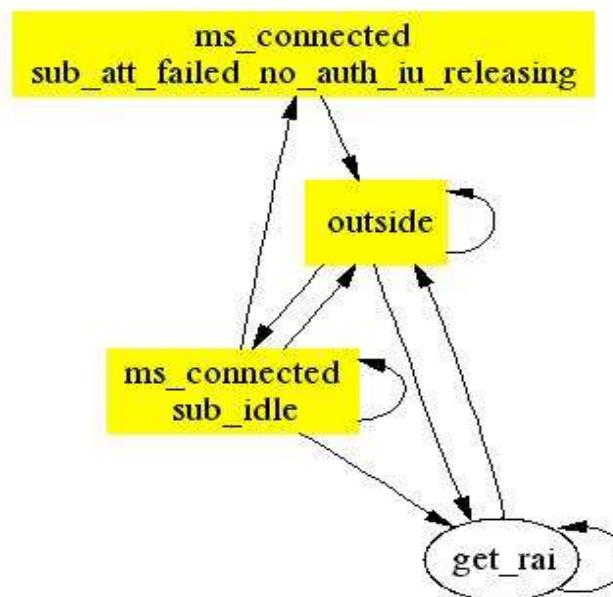
After filtering and abstraction we can obtain a trace like:

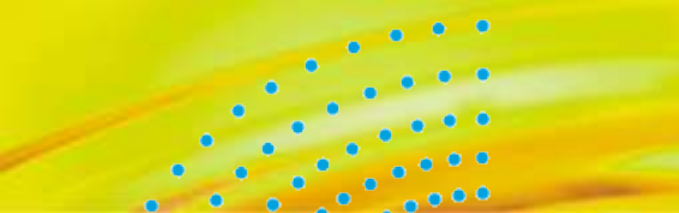
```
[{state,outside},...,{state,outside},  
 {state,[ms_connected,sub_idle]},  
 {event,get_rai},  
 {state,outside},...,{state,outside},  
 {state,[ms_connected,sub_idle]},  
 {event,get_rai},  
 {state,outside},...,{state,outside},  
 {state,[ms_connected,sub_idle]},  
 {state,[ms_connected,sub_idle]},  
 {state,outside},...,{state,outside},  
 {event,get_rai},  
 ...]
```

Structuring trace data

Represented as a graph, this gives a clear picture of possible scenarios

```
[{state,outside},...,{state,outside},
 {state,[ms_connected,sub_idle]},
 {event,get_rai},
 {state,outside},...,{state,outside},
 {state,[ms_connected,sub_idle]},
 {event,get_rai},
 {state,outside},...,{state,outside},
 {state,[ms_connected,sub_idle]},
 {state,[ms_connected,sub_idle]},
 {state,outside},...,{state,outside},
 {event,get_rai},
 ...]
```





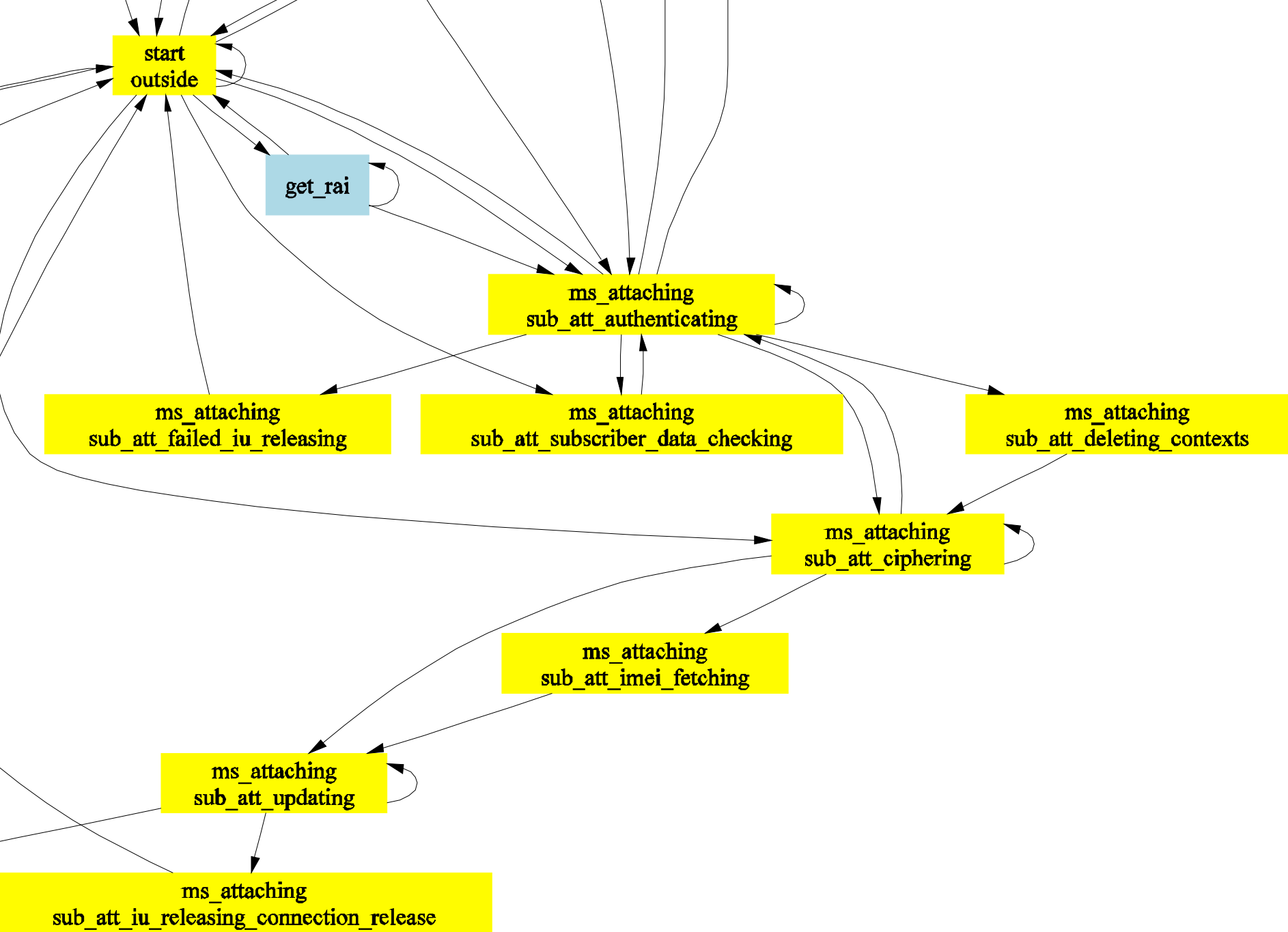
Comparison

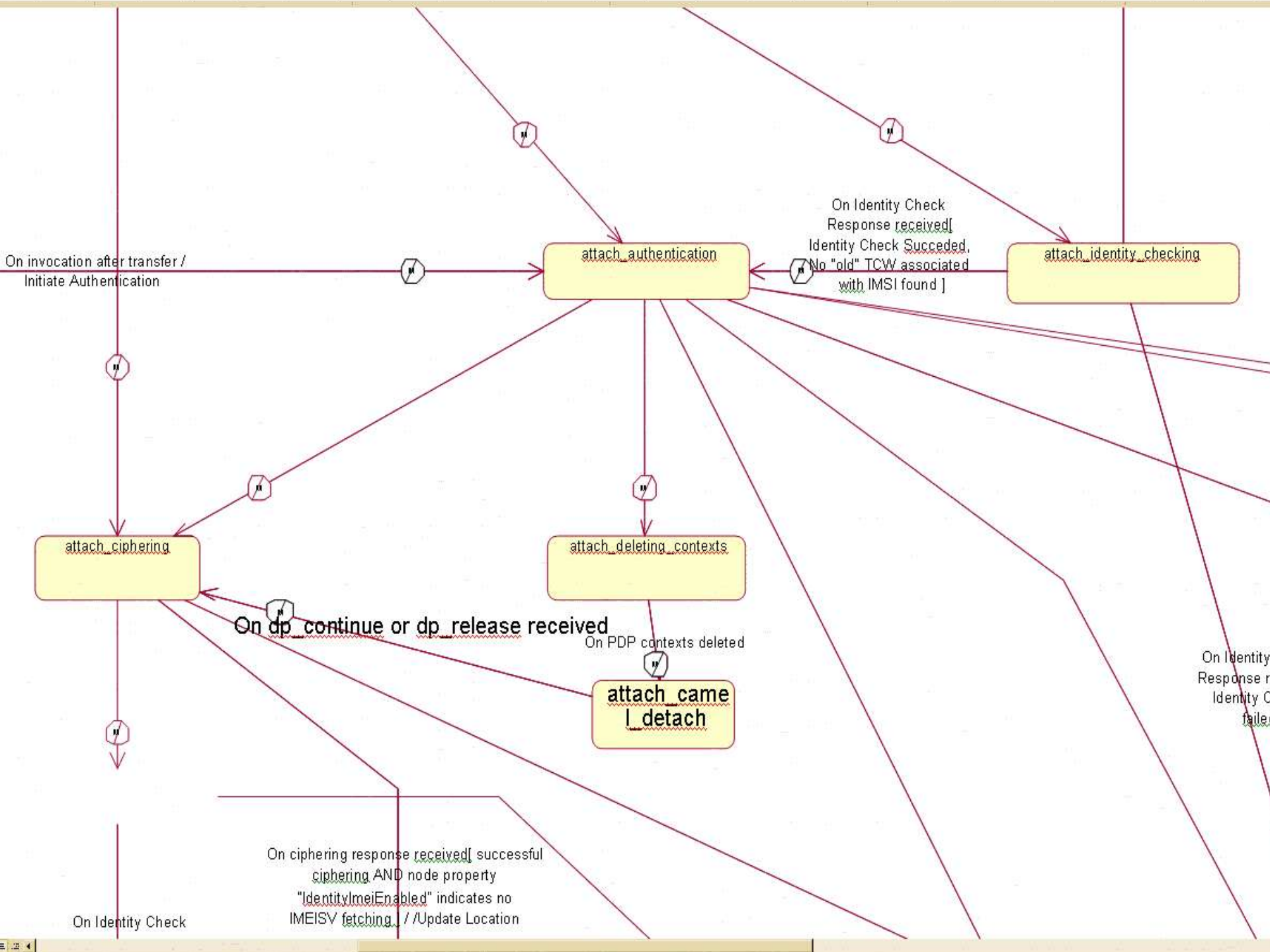
We found (after manual mapping of names):

States in obtained diagram that are not in original

State transitions in obtained diagram that are not in original

States and transitions in original that do not show in obtained diagram







Future work

- *Analyze more blocks*
- *Connect Rational visualization tools to our analysis (show differences in one picture)*
- • *Automatically generate test cases such that most states and events in the original are covered in the trace*
- *Generate Hierarchical State Machine model*