Using Model Learning for the Generation of Mock Components

Sébastien Salva
LIMOS, Clermont Auvergne University, France

32nd IFIP International Conference on Testing Software and Systems, dec. 2020

Who am I?

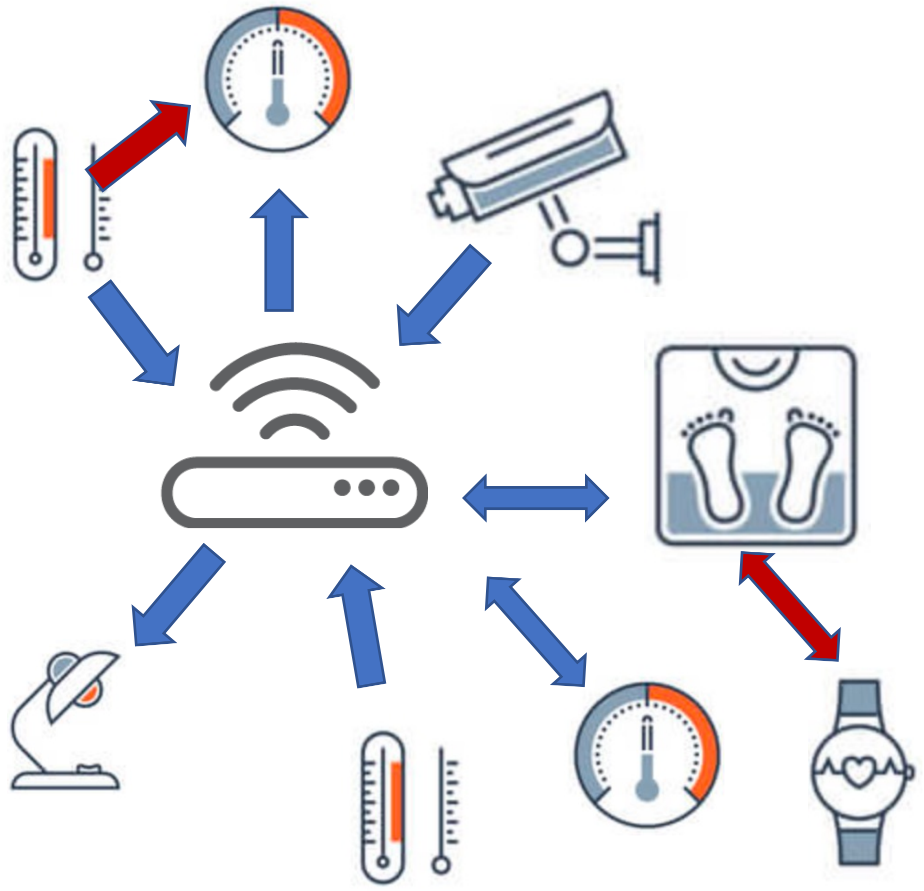
```
Public void setUp(){
Identity id=new Identity(“salva’’);}

Public void testid (){
assertEquals(id.surname, ’’sébastien’’);
assertEquals(id.name, ’’salva’’);
assertEquals(id.labo, ’’LIMOS’’);
assertEquals(id.univ ’’University Clermont Auvergne, France’’);

assertArrayEquals(i.recherche, new String[] {“testing’’, “security’’, ’’model learning’’,
“services“});
}
```

Paper presentation

- **Using Model Learning for the Generation of Mock Components**
  (in the context of IoT systems)

  1. Overview
  2. Model learning from logs of IOT systems
  3. Choice of the components to mock
  4. Mock generation/execution
  5. (preliminary) Evaluation

Issues



IoT systems

- made up of heterogeneous components
- a lot of (complex) dependencies

=> Difficult to test, to write test cases

Uncontrollable components

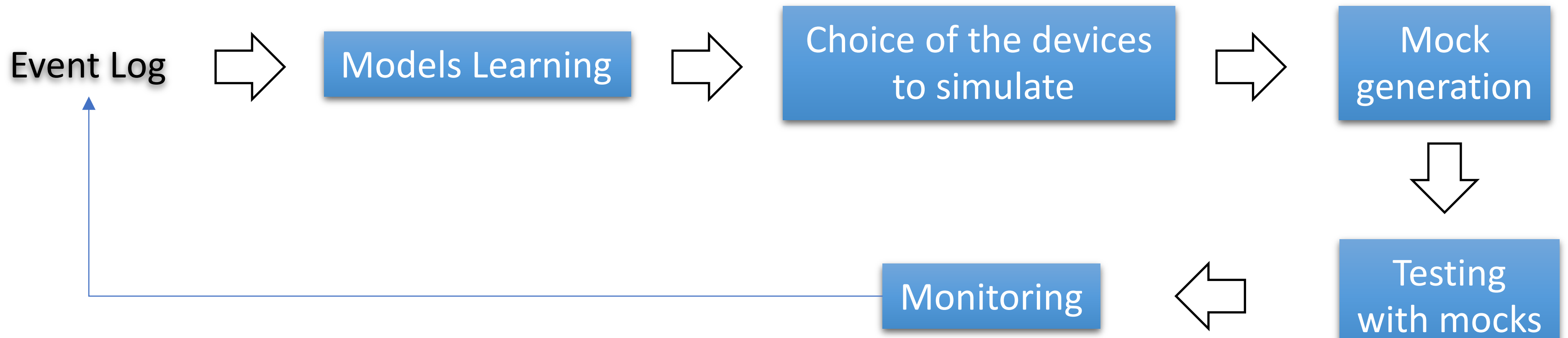
=>prevent to cover all the behaviours

Simulating/replacing some components with mocks usually helps

- must be manually developed
- Difficult to maintain mocks when system is updated

Approach Goals

- Generate Stubs/mocks allowing to simulate real IoT devices
  - For replacing complex dependencies
  - For replacing untestable devices (uncontrollable)
- Help choose the components to replace

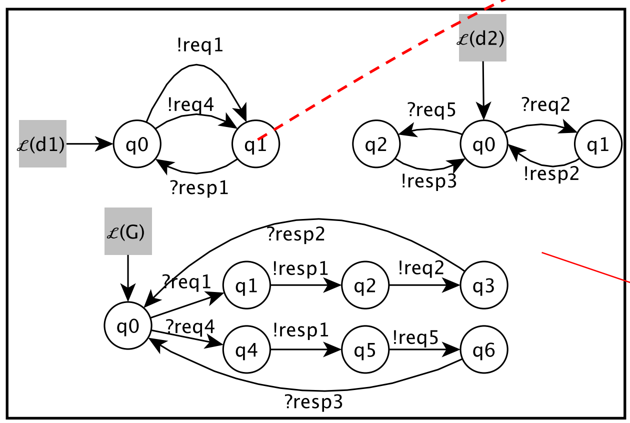


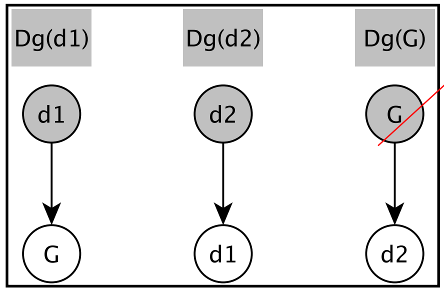
Big Picture

Event Log ⇨ Models Learning ⇨ Choice of the devices to simulate ⇨ Mock model generation ⇨ Testing with mocks

Event LOG

```
09:39:53.416
req1(from:=d1,to:=G,param:=udevice,nvalue:=64,svalue=TEMP)
09:39:53.848
resp1(from:=G,to:=d1,content:=ok)<br>
09:39:54.216
req2(from:=G,to:=d2,type:=command,nvalue:=64,svalue=TEMP)
09:39:55.429
resp2(from:=d2,to:=G,content:=received 64)
...
```

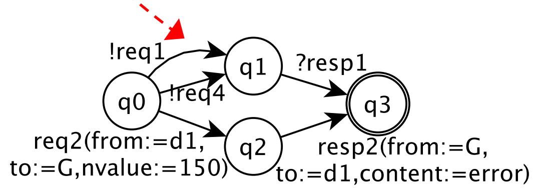




Quality metric evaluation (testability, dependability , etc.)

⇩

Choose



!req1
!req4
q0
q1
q2
q3
?resp1
req2(from:=d1, to:=G,nvalue:=150)
resp2(from:=G, to:=d1,content:=error)

Mock Runner

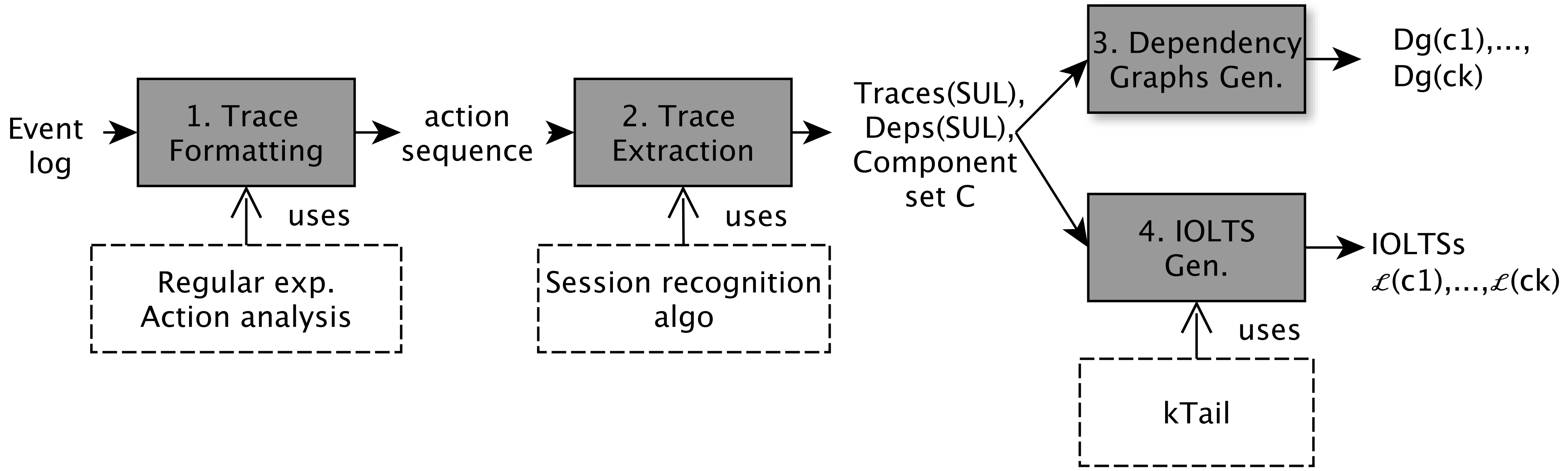
Model learning from logs of IOT systems

Assumptions :

1. Log with events, timestamps,
2. Component identified, recognise request/response, non communication event
3. (Session recognition)
    I. Session id., or
    II. requests processed on a first-come, first served basis, request-response exchange pattern (req-> 1 or multiple resp)

Model learning from logs of IOT systems

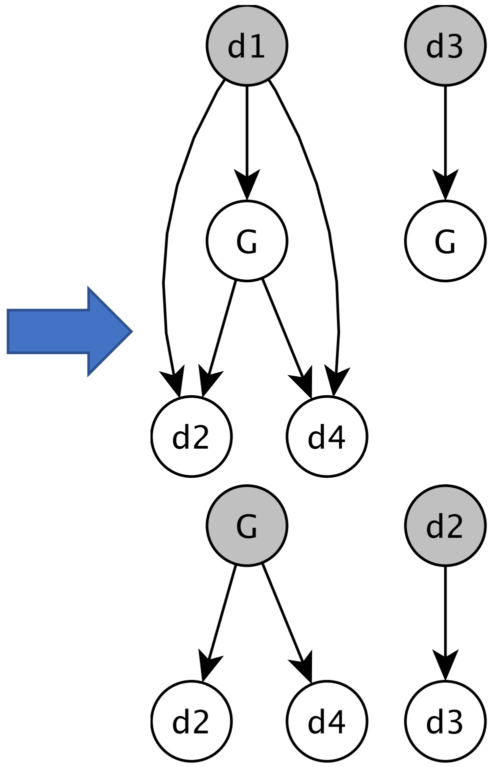
CkTail (Comunicating system kTail) Algorithm :

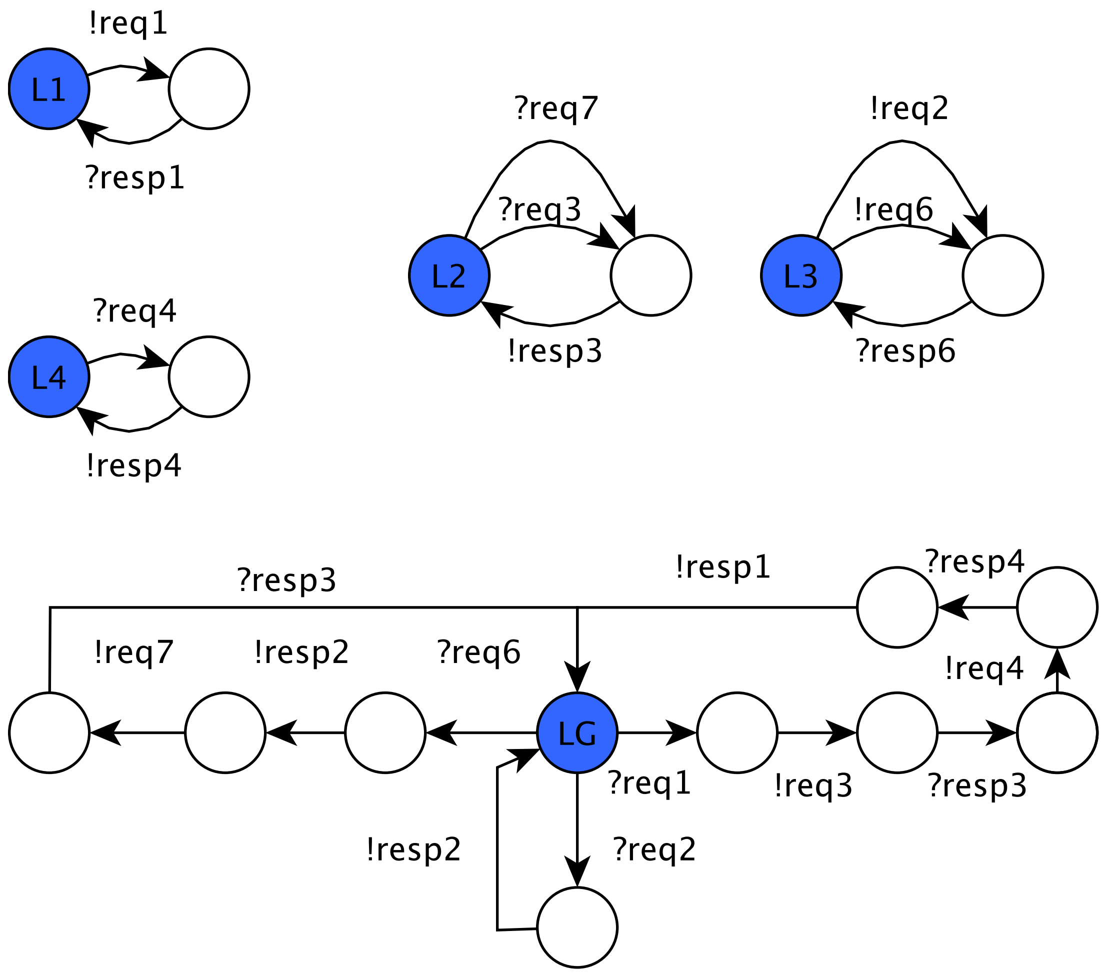


Model learning from logs of IOT systems

Action seq.

```
09:39:53.416
req1(from:=d1,to:=G,param:=udevice,svalue:=open)
09:39:53.848
req2(from:=d3,to:=G,param:=udevice,svalue:=66)
09:39:55.416
req3(from:=G,to:=d2,param:=heating,cmd:=On)
09:39:55.429 resp3(from:=d2,to:=G,switchcmd:=done)
09:39:55.430
req4(from:=G,to:=d4,param:=heating,cmd:=On)
09:39:55.433
resp4(from:=d4,to:=G,switchcmd:=done)<br>
09:39:55.567 resp1(from:=G,to:=d1,content:=req sent)
09:39:55.629 resp2(from:=G,to:=d3,content:=ok)
09:44:19.714
req6(from:=d3,to:=G,param:=udevice,svalue:=68)
09:44:19.727 resp2(from:=G,to:=d3,content:=ok)
09:44:19.727
req7(from:=G,to:=d2,param:=heating,cmd:=Off,svalue:=68
)
09:44:19.866 resp3(from:=d2,to:=G,switchcmd:=done)
```





Choice of the components to mock

- Could be manually performed but … seems difficult

- Evaluation of models with 6 quality metrics
  - Understandability
  - Accessibility
  - Testability (obs, cont)
  - Dependability (in-deps, out-deps)

Choice of the components to mock

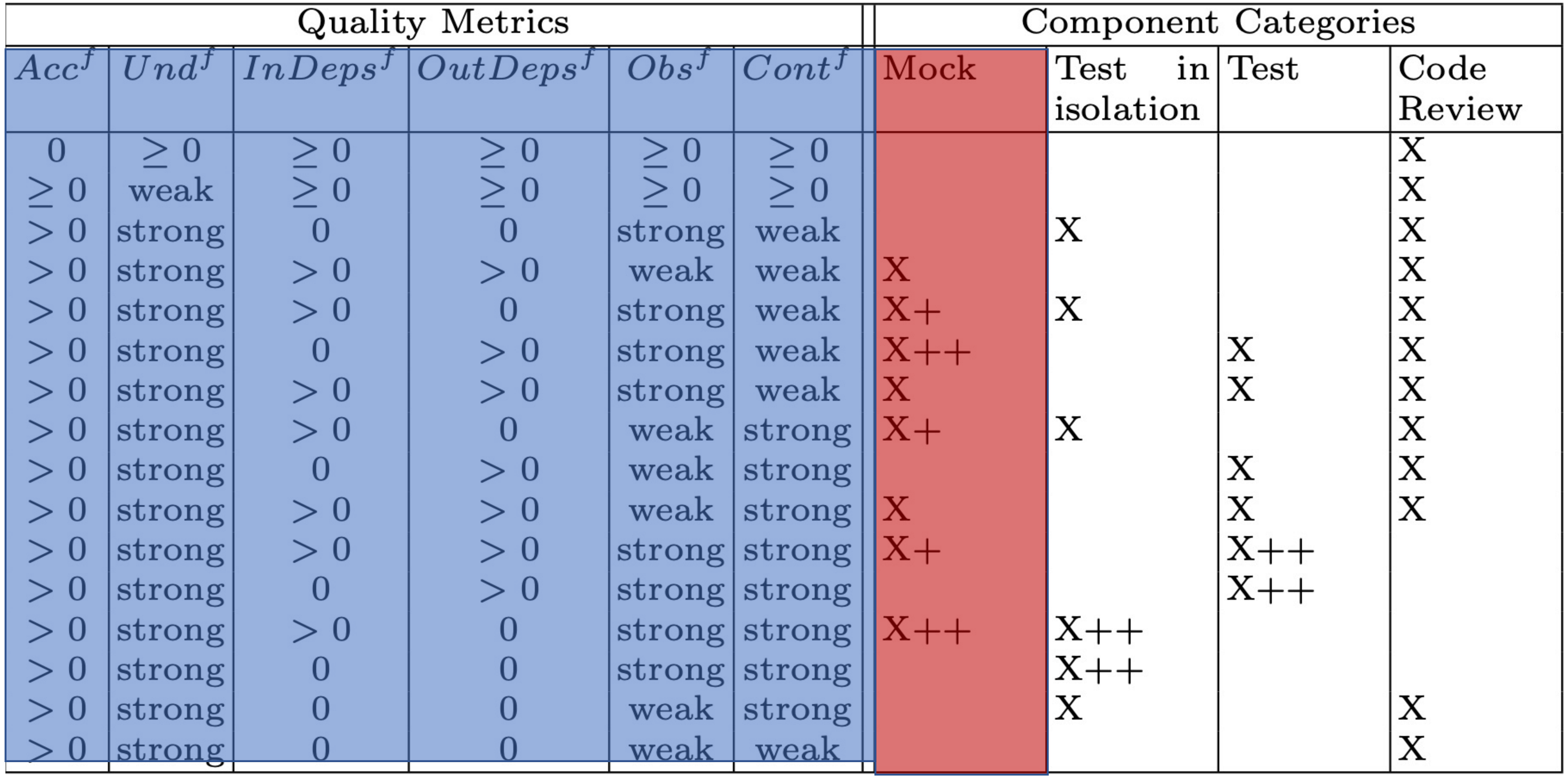
- Metrics have to be interpreted for choosing the components to mock

- => proposition of a metric interpretation

- Classification of Components into categories
  - Testable, Testable in isolation, mockable, untestable (code review)

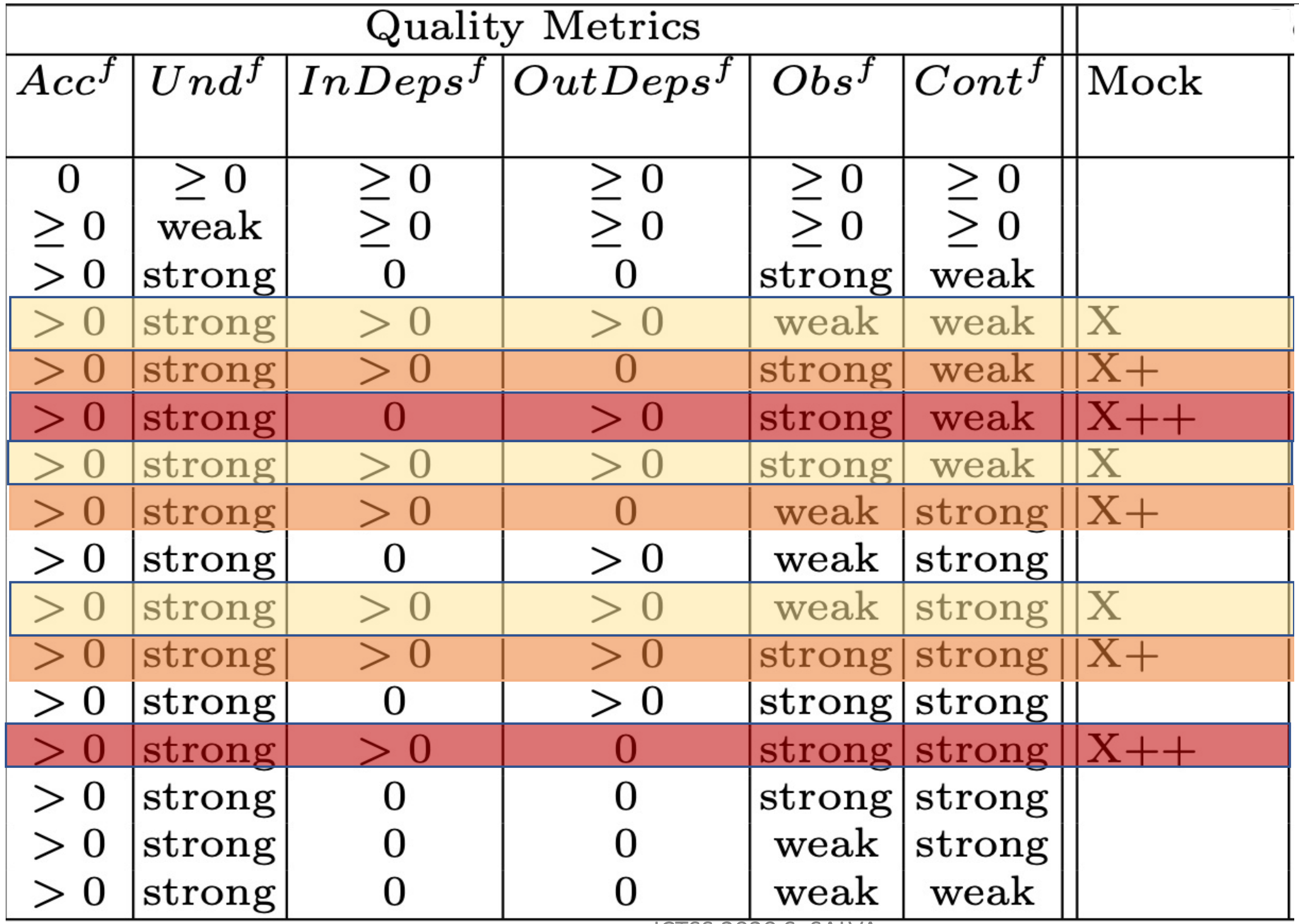
- We propose 3 levels of interest

Choice of the components to mock



Quality Metrics						Component Categories			
Acc^f	Und^f	$InDeps^f$	$OutDeps^f$	Obs^f	$Cont^f$	Mock	Test in isolation	Test	Code Review
0	≥ 0	≥ 0	≥ 0	≥ 0	≥ 0				X
≥ 0	weak	≥ 0	≥ 0	≥ 0	≥ 0				X
> 0	strong	0	0	strong	weak		X		X
> 0	strong	> 0	> 0	weak	weak	X			X
> 0	strong	> 0	0	strong	weak	X+	X		X
> 0	strong	0	> 0	strong	weak	X++		X	X
> 0	strong	> 0	> 0	strong	weak	X		X	X
> 0	strong	> 0	0	weak	strong	X+	X		X
> 0	strong	0	> 0	weak	strong			X	X
> 0	strong	> 0	> 0	weak	strong	X		X	X
> 0	strong	> 0	> 0	strong	strong	X+		X++	
> 0	strong	0	> 0	strong	strong			X++	
> 0	strong	> 0	0	strong	strong	X++	X++		
> 0	strong	0	0	strong	strong		X++		
> 0	strong	0	0	weak	strong		X		X
> 0	strong	0	0	weak	weak				X

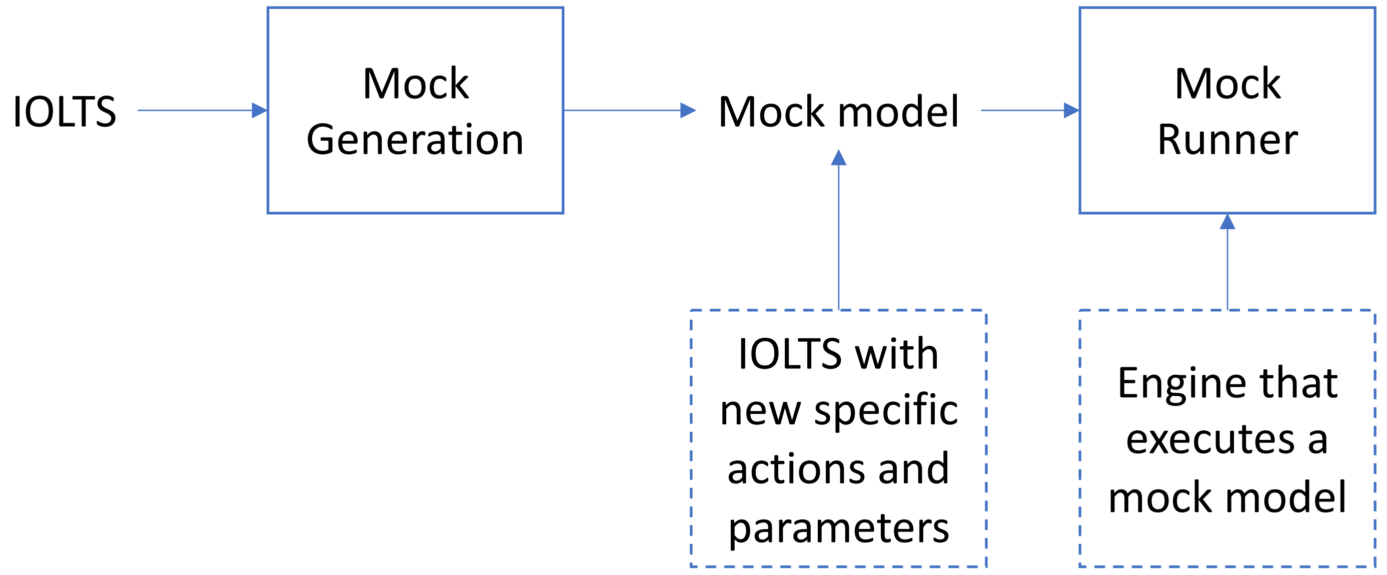
Choice of the components to mock



Quality Metrics						
Acc^f	Und^f	$InDeps^f$	$OutDeps^f$	Obs^f	$Cont^f$	Mock
0	≥ 0	≥ 0	≥ 0	≥ 0	≥ 0	
≥ 0	weak	≥ 0	≥ 0	≥ 0	≥ 0	
> 0	strong	0	0	strong	weak	
> 0	strong	> 0	> 0	weak	weak	X
> 0	strong	> 0	0	strong	weak	X+
> 0	strong	0	> 0	strong	weak	X++
> 0	strong	> 0	> 0	strong	weak	X
> 0	strong	> 0	0	weak	strong	X+
> 0	strong	0	> 0	weak	strong	
> 0	strong	> 0	> 0	weak	strong	X
> 0	strong	> 0	> 0	strong	strong	X+
> 0	strong	0	> 0	strong	strong	
> 0	strong	> 0	0	strong	strong	X++
> 0	strong	0	0	strong	strong	
> 0	strong	0	0	weak	strong	
> 0	strong	0	0	weak	weak	

Mock generation/execution

Mock = Mock model, which is executed by a Mock Runner


IOLTS
Mock Generation
Mock model
Mock Runner
IOLTS with new specific actions and parameters
Engine that executes a mock model

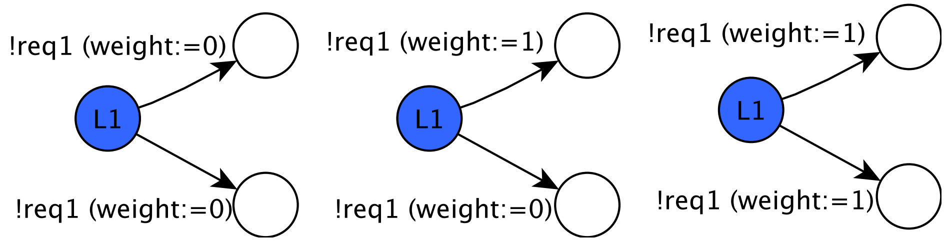
Mock model
- Written by hands from scratch (difficult)
- Transform the inferred IOLTSs

Mock generation/execution

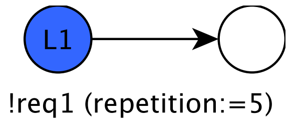
Mock model

IOLTS with some specific parameters:

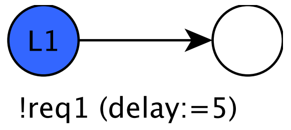
weight:

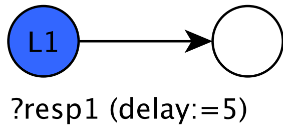


repetition:



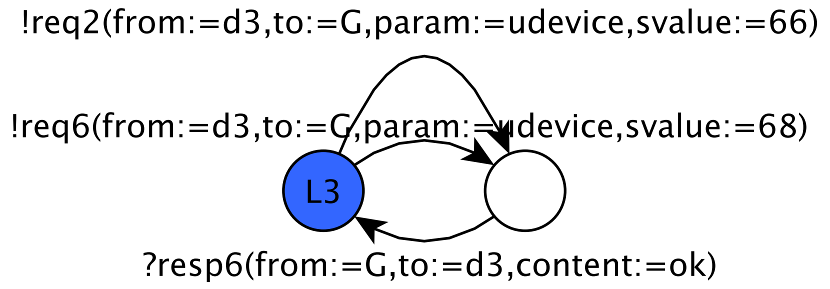
delay:



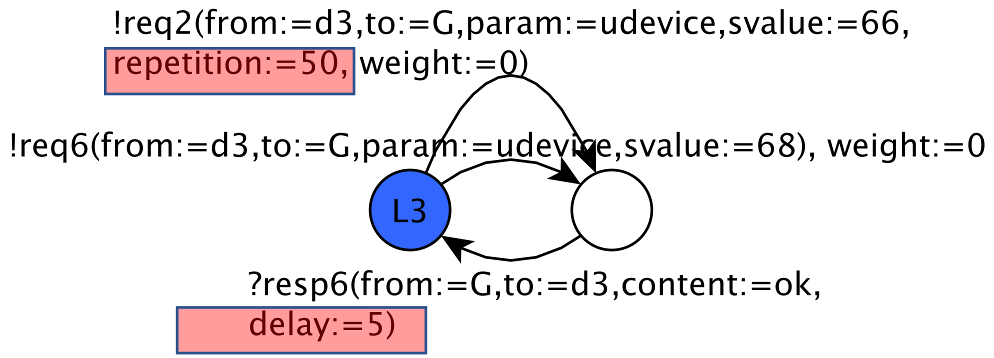


Mock generation/execution

Mock model examples:


!req2(from:=d3,to:=G,param:=udevice,svalue:=66)
!req6(from:=d3,to:=G,param:=udevice,svalue:=68)
L3
?resp6(from:=G,to:=d3,content:=ok)

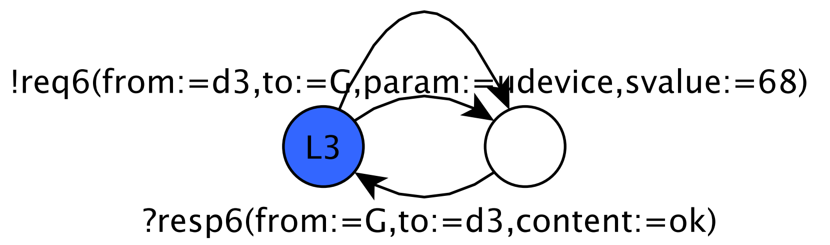
Original IOLTS


!req2(from:=d3,to:=G,param:=udevice,svalue:=66, repetition:=50, weight:=0)
!req6(from:=d3,to:=G,param:=udevice,svalue:=68), weight:=0
L3
?resp6(from:=G,to:=d3,content:=ok, delay:=5)

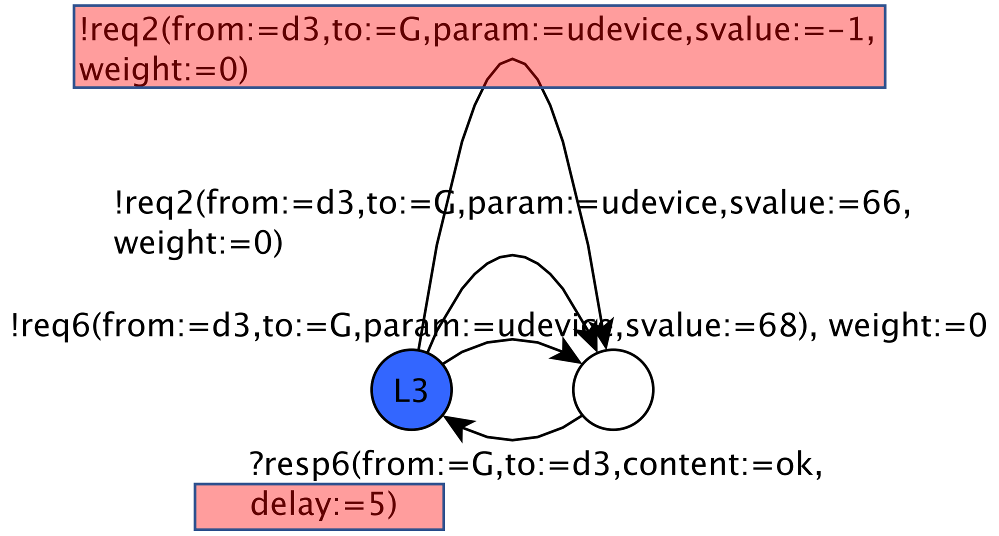
Mock model
(repetition of req2, delay for ?resp6)

Mock generation/execution

Mock model examples:


!req2(from:=d3,to:=G,param:=udevice,svalue:=66)
!req6(from:=d3,to:=G,param:=udevice,svalue:=68)
L3
?resp6(from:=G,to:=d3,content:=ok)

Original IOLTS


!req2(from:=d3,to:=G,param:=udevice,svalue:=-1, weight:=0)
!req2(from:=d3,to:=G,param:=udevice,svalue:=66, weight:=0)
!req6(from:=d3,to:=G,param:=udevice,svalue:=68), weight:=0
L3
?resp6(from:=G,to:=d3,content:=ok, delay:=5)

Mock model
(new action, delay for ?resp6)

Mock generation/execution

Mock Runner ~ Engine that executes a mock model

- Implemented as a Rest service
- Performs concrete executions by following the paths of a mock model
  - Starts from q0 and either waits for an input or executes an ouput
  - Builds and stores runs (alternate sequence of states and actions)
  - If Mock runner receives an unexpected action, it returns an error in its log (can be used by testers)
  - Stops a current execution in an terminal or deadlock state

(preliminary) Evaluation

Perfomed from a real IoT system (20 devices, 2 gateways)

Conducted with 24 students (fourth-year in Computer Science)

- Measured the times required to code mocks from logs from scratch and times for generating mocks
  - Approach provides greater efficiency than building mocks by hands
  - cuts the time by 75%

- Our mocks can replace reals devices if the inferred IOLTS are precise enough precise (no over- or under-approximated)

Limitations

- Mock runner can be called from test cases but in a limited way at the moment
  - (start mock runner, give mock model, get runs and errors)
  - -> could be extended (get number of inputs received?, outputs sent ?)

- Current Mock runner impl. requires some resources (mem, and cpu),
  - ok if deployed on Web servers

- Choice of mockable devices made from the interpretation of metrics.
  - depends on the systems ? Or the dev. Tools etc.

Perspectives

- Propose a better solution than a tabular of rules for choosing the devices that can be replaced by mocks
- Executions with natural flows of parameter values (with maps instead of values)
- Automatic generation of mock models from IOLTSs with different strategies
  - Strategy for robustness testing (!nject unexpected events, etc.)
  - Strategy for security testing (inject malicisous behaviours, etc.)

Thanks

- Questions ?

Assumptions

A1 Event log: black-boxes The communications among the com- ponents can be monitored, Event logs are collected in a synchronous environment made up of synchronous communications. Besides, these events are ordered by means of timestamps given by a global clock.

A2 Event content: components produce communication events or non-communication events. iden- tify the source and the destination of each event. a communication event can be iden- tified either as a request or a response;

A3 Device collaboration: components can run in par- allel and communicate with each other.
– A32: the events that belong to the same session are identified by a parameter assignment.

– A31: they cannot run multiple instances; requests are processed by a component on a first-come, first served basis. Besides, components follow the request –response exchange pattern (a response is associated to one request, a request is associated to one or more responses),

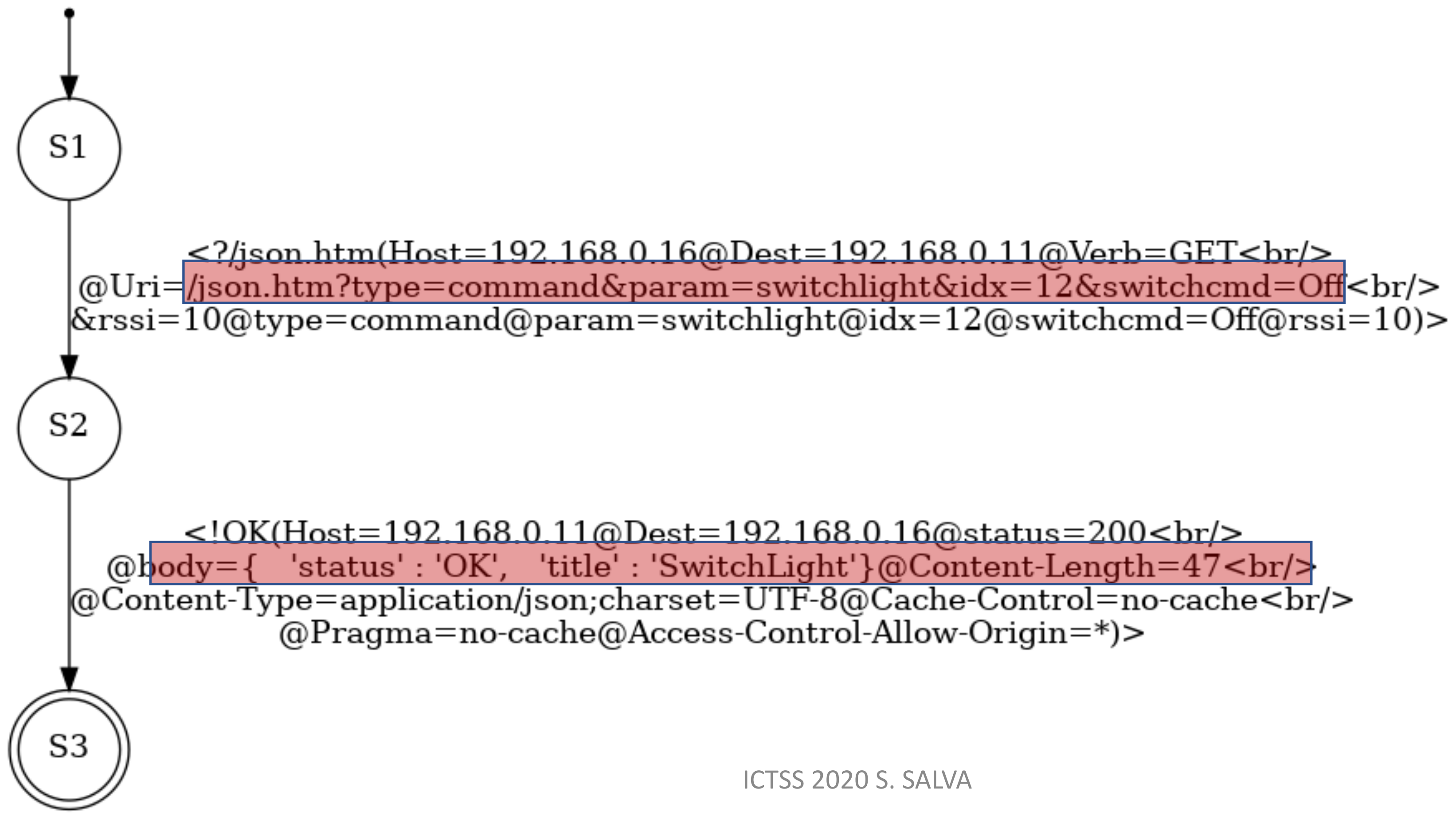
Mock generation/execution

A good mock should be compliant with :

component that mimics the behaviours of another real component (H1).
A mocks should be easily created, easily set up, and directly queriable (H2).
In the tests, the developer has to specify how the mock ought to be exercised (H3).
Besides, a mock can be handled by tests to verify that it runs as expected (H4).
If the mock is not exercised as expected, it should return an error so that tests fail (H5).

Mock generation/execution

Mock model example:



```
<?/json.htm(Host=192.168.0.16@Dest=192.168.0.11@Verb=GET<br/>
@Uri=/json.htm?type=command&param=switchlight&idx=12&switchcmd=Off<br/>
&rssi=10@type=command@param=switchlight@idx=12@switchcmd=Off@rssi=10)>
```

```
<!OK(Host=192.168.0.11@Dest=192.168.0.16@status=200<br/>
@body={   'status' : 'OK',   'title' : 'SwitchLight'}@Content-Length=47<br/>
@Content-Type=application/json;charset=UTF-8@Cache-Control=no-cache<br/>
@Pragma=no-cache@Access-Control-Allow-Origin=*)>
```