EVENT-BASED ANALYSIS OF REAL-TIME ACTOR MODELS

Haukur Kristinsson

Reykjavik University

August, 2012

What is involved in this project?

- Modeling and reasoning about real-time distributed systems.
- Verifying correctness properties.
- Validating models using performance analysis with event-based simulation.
  - Response Time Analysis
  - Frequency Analysis
  - Quantitative Analysis
- We show the applicability of our methods using examples and case studies.

Distributed and asynchronous systems

- This project focuses on networked applications (specifically with distributed and asynchronous patterns)
- What are distributed and asynchronous systems?
  - Web services
  - Concurrent sensor networks
  - Modern messaging

Actor Model

- Mathematical model of concurrent computation.
- Treats *actors* as primitives of concurrent computation that can:
  - Send messages to other actors.
  - Create of new actors.
  - Determine how to respond to the next message received.
- All actors run concurrently in a system.
- Uses asynchronous message passing for communication.
- Simple and easy to reason about.

How do we do analysis for such systems?

- Follow model-driven engineering guidelines.
- Based on modelling the abstract behaviour of the system.
- Mathematical precise models, similar to other engineering principles.
- Two highlights for analyzing models:
  - Verification that ensures that a specification is satisfied.
    - Is the model correct? (errors or failure).
  - Analysis that ensures intended performance requirements are met in terms of the methods employed and the result obtained.
    - Is the system built right? (performing as promised)

Rebeca

Reactive Objects Language, Rebeca, is an operational interpretation of the actor model with formal semantics and model checking tools.

- Based on the **Actor model**
- Formal semantics
- Reactive objects (rebecs)
- Easy to use by software engineers
  - Message-driven and object-based computational model.
  - Java-like syntax.
  - Set of verification tools.

Rebeca Code...

```
reactiveclass ReceiverObj {
  knownrebecs {
    SenderObj senderObj;
  }
  msgsrv send() {
    int t = ?(-1,1);
    if (t != -1) {
      senderObj.ack();
    }
  }
}
```

Listing 1: Reactive class ReceiverObj

```
main {
  ReceiverObj
       receiverObj(senderObj):();
  SenderObj
       senderObj(receiverObj):();
}
```

Listing 2: receiverObj instantiated with reactive class ReceiverObj.

Timed Rebeca

Timed Rebeca is an extension to Rebeca with real-time features.

- Computation time modeled with **delay**
  - Time progresses locally when rebec calls delay().
- Message delivery time with **after**
  - Message sends are tagged with time of send.
- Message expiration with **deadline**
  - Time progresses when a rebec picks up a message to execute.
- Periodic event with a *message loop* and **after**.

Time constructs

- **Delay**: delay(t), increase the value of the local clock of the respective rebec by the amount t.
- **Now**: now(), returns the time of the local clock of the rebec from which it is called.
- **Deadline**: r.m() deadline(t), where r denotes a rebec name, m denotes a method name of r and t is a natural number.
- **After**: r.m() after(t), where r denotes a rebec name, m denotes a method name of r and t is a natural number.

What do we use as our analysis back-end?

Erlang

Erlang is a functional programming language for programming real-time distributed systems.

- Based on the Actor model.
- We translate Timed Rebeca models to Erlang.

McErlang

McErlang is a model checker that uses an Erlang program as the model.

- Replaces Native Erlang run-time engine with a new one.
- Is able to capture all program states.
  - Taking into account the distribution, communication and message boxes.
- Can construct a complete state-space of an Erlang program.
- Has on-the-fly verification algorithms.
- Until now it had no real-time semantics.

The project is based on existing translation to Erlang and McErlang (Aceto et al., 2011).

- Automated translation of Timed Rebeca to Erlang
- Only simulation (execution manner).
  - Depended on the system clock.
  - Timings could be ambiguous.
- No verification ability.
- Simulation run-time monitoring via McErlang.

We wanted to get more from our language:

- More variety of analysis techniques.
  - Verification
  - Visualization of simulations
  - Performance Analysis
- Be able to model larger set of behaviors.
  - More variety of data-types.
  - Custom functions for special logic.
- Add tracing capability to models.
  - Checkpoints
  - Events

Extensions introduced

- Lists - **list**<**int**> *var*
  - Mandatory for modeling queues or buffers in systems.
- Custom functions - **erlang**.<*func*>(*parameters*)
  - Write function in Erlang and use within the model.
- Checkpoint - **checkpoint**(*label*(,*terms*))
  - Keeping track of interesting points in the model when *simulating*.
  - Exposing Timed Rebeca variables from the program state-space when *verifying*.

Extensions introduced

Extension	Timed Rebeca Syntax		Erlang / McErlang
Lists	**list**$< int > N$;	$\rightarrow$	Erlang list data type as a variable with name N.
Custom Function	**erlang**.*func*$(V_1,...,V_n)$;	$\rightarrow$	Call to function *func* with parameters $V_1,...,V_n$.
Checkpoints	**checkpoint**$(L, T(, T_1,...,T_n))$;	$\rightarrow$	Erlang Output Function with L as the label of the checkpoint, T as the term and $T_1,...,T_n$ as a optional tuple of term data which is used when doing simulation analysis.
	***checkpoint**$(L, T(, T_2,...,T_n))$;	$\rightarrow$	McErlang probe with L as the label of the checkpoint and T as the term. *Additional parameters are extending the term to a tuple of data.*

Timed Rebeca and McErlang's Timed Semantics

In cooperation with Lars-Ake Fredlund (creator of McErlang) we extended the mapping to use with new timed semantics for McErlang.

- Explicit-Time Approach, where tick rules are defined in timeouts in Erlang.
- We disabled the usage of system time (now()).
- We use clock references for expiration of messages.
  - McErlang API: Ref = *now*(), *nowRef*(), *was*(Ref), and *forget*(Ref).

Support for the timed semantics

Erlang concurrently spawning two processes

ProcccessOne = spawn(process, function, [1]),
ProcccessTwo = spawn(process, function, [2]).

In Erlang:

- Nothing that prevents ProcessOne to run faster than ProcccessTwo.
- McErlang checks for all possible executions.
- No order of execution respected based on delays.

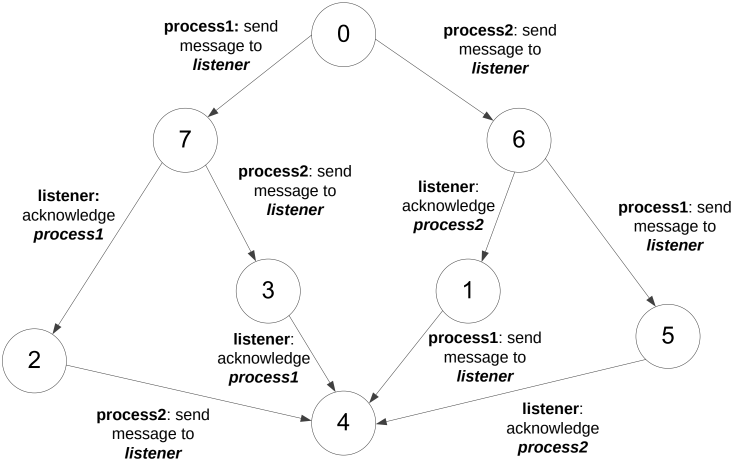
Timed Rebeca model - Competing processes:

```
reactiveclass Process(2) {
  knownrebecs { Listener lsnr; }
  statevars { }

 msgsrv initial(int delaybeforesend) {
  delay(delaybeforesend);
  lsnr.receiveMsg(delaybeforesend);
 }

 msgsrv ack(boolean processed) {
  checkpoint(isProcessProcessed,
processed);
 }
}
```

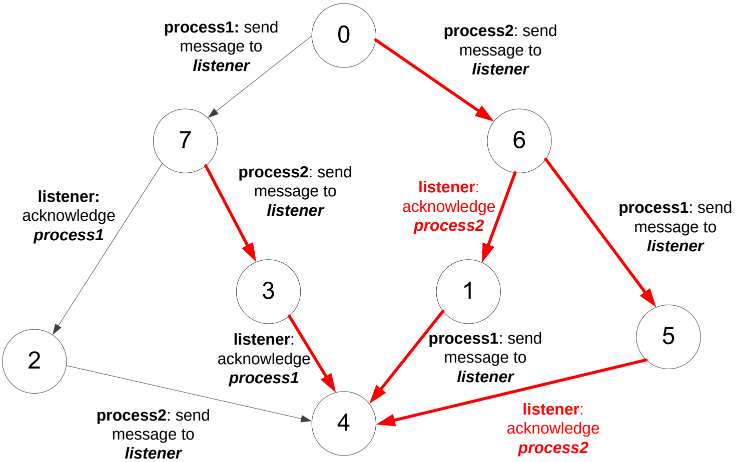
```
reactiveclass Listener(2) {
  knownrebecs { Process proc1; Process proc2; }
  statevars { boolean received; }

  msgsrv initial() { received = false; }

  msgsrv.receiveMsg(int delaylabel)
  {
   // Note: Only receives once
   if(received == false) {
    received = true;
    sender.ack(true);
   }
  }
}
```

```
main {
  Listener listener(process1,process2):(); % Listener that only processes one message
  Process process1(lsnr):(2); % Delays by 2 before sending
  Process process2(lsnr):(3); % Delays by 3 before sending
}
```

McErlang is a model checker for Erlang which has operational semantics that differ from Timed Rebeca's.



Transitions marked as red shouldn't happen.

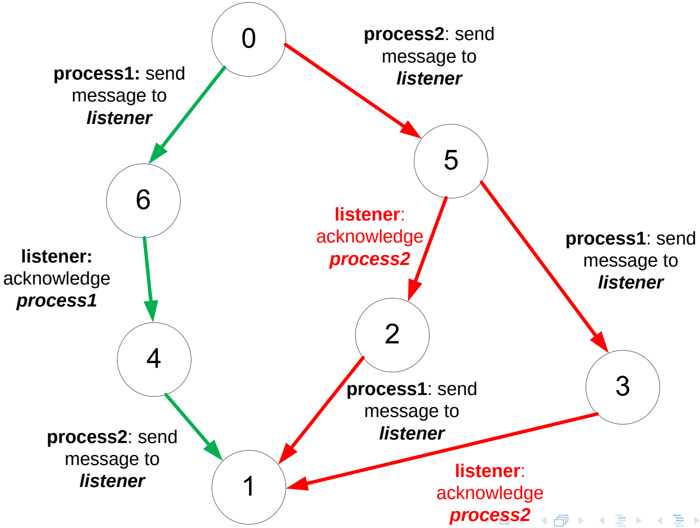


Support for the timed semantics

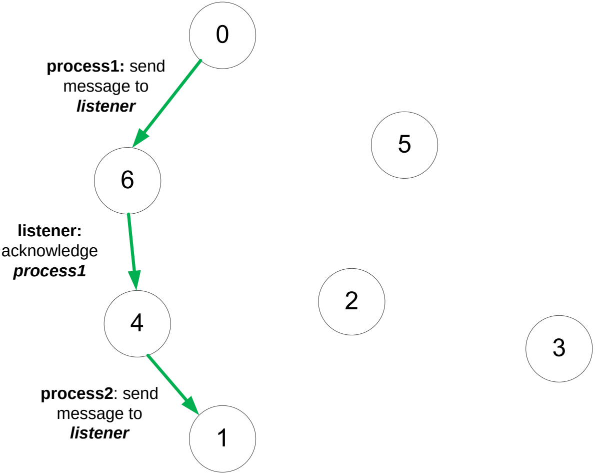
Timed Rebeca message send is translated with or without timeout in Erlang.

- Define non-timed actions, as *urgent* with maximum timeout as 0.
- New timed semantics allow us to mark timed actions also as *urgent*.
  - Orders urgent transitions based on their timeouts.
  - Define maximum delay time as the timeout.

Non-timed actions, defined as *urgent* with maximum timeout as 0:



Timed actions, also defined as *urgent* with maximum delay time as the timeout:



McErlang Safety monitors

- Acts like a safety property.
- Can observe each generated program state by McErlang.
  - on-the-fly verification.
  - violates or satisfies states in lock-step manner.
- Can access information from program states:
  - process mailboxes
  - status of the processes
  - process dictionaries
  - process actions (sending or receiving)
- Pre-defined monitors
  - Deadlock detection
  - Process queue detection

Example of a deadlock monitor

```
monitorType() -> safety.

init(State) -> {ok,State}.

stateChange(State,MonState,_) ->
 case is_deadlocked(State) of
  true -> deadlock;
  false -> {ok, MonState}
 end.

 is_deadlocked(State) ->
 State#state.ether =:= [] andalso case mce_erl:allProcesses(State) of
  [] -> false;
  Processes ->
  case mce_utils:find(fun (P) -> P#process.status =/= blocked end, Processes) of
   {ok, _} -> false;
   no -> true
  end
end.
```

Example of a deadlocking model

```
reactiveclass DeadlockingExample(3) {
 knownrebecs { DeadlockingExample proc; }
 statevars { }

 msgsrv initial() {
   proc.go();
 }

 msgsrv go()
 {
    /* Random pick of integer */
    int rand = ?(1,2,3,4,5);
    delay(1);

    if(rand != 5) {
        sender.go();
    }
  }
}

main {
 DeadlockingExample proc1(proc2):();
 DeadlockingExample proc2(proc1):();
}
```

Violation after 0:010 runtime seconds!
120 states and 280 transitions
Process was halted showing a deadlock

Example of a deadlock-free model

```
reactiveclass DeadlockingExample(3) {
 knownrebecs { DeadlockingExample proc; }
 statevars { }

 msgsrv initial() {
   proc.go();
 }

 msgsrv go()
 {
    /* Random pick of integer */
    int rand = ?(1,2,3,4,5);
    delay(1);

    if(rand != 6) {
        sender.go();
    }
  }
}

main {
 DeadlockingExample proc1(proc2):();
 DeadlockingExample proc2(proc1):();
}
```

Satisfied after 0:210 runtime seconds!
444 states and 818 transitions
No process showed a deadlock in any state.

Safety Checking with **checkpoints**

```
...
checkpoint(id1,exp1);
checkpoint(id2,exp2,exp3);
...
```

→

```
...
mce_erl:probe(id1,exp1),
mce_erl:probe(id2,{exp2,exp3}),
...
```

With probes we can expose program variables to be used with McErlang *monitors*.

- No well-defined mechanism for retrieving value of program variables from the program state.
- Probes *term* (values) are easily accessible from the program state based on their *labels*.

McErlang Monitor: Property based on the checkpoint with the label "id1"

```
% Check if state has any checkpoints with the Label
case has_probe_with_label(id1,StateActions) of
  {true,RetrievedTerm} -> ManipulateTerm(RetrievedTerm);
  false -> {ok,noCheckPointFoundWithLabel}
end.

% Function that does something with the retrived term
ManipulateTerm(TermRetrived) ->
  case TermRetrived == 5 of
     % Violate if term is equal to 5.
     true -> {violation_termIsFive,TermRetrieved};
     % Satisfy term if not 5.
     false -> {ok,satisfied}
  end.
```

Writing properties based on checkpoints

- Hard to write monitors without Erlang knowledge.
- Future work: Automated generation of monitors with LTL support.
- Timed Rebeca monitor:
  - Check for dropped messages.
  - Check if a checkpoint with a label occurs.
  - Compare checkpoint term with an integer or boolean.

Discrete-Event Simulation (DES) Approach

- DES represents the operation of a system as a chronological sequences of events.
- Each event occurs at a certain time.
- In this project we define these events when:
  - Message is taken from a rebec message bag.
  - When a checkpoint occurs.
- Generated data from a model can contain multiple simulation.
- Each simulation contains multiple traces (events and checkpoints).
- Implemented a tool-set that supports analysis of the generated data.

McErlang Timed Semantics and Simulation

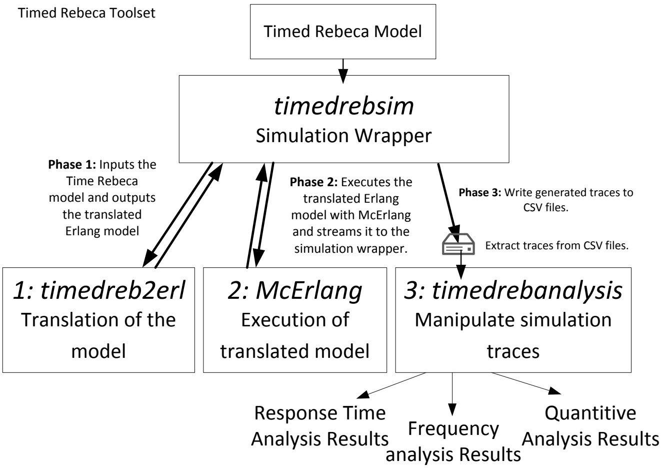
- Use a different exploration algorithm.
  - Chooses next transition randomly.
- Time passage corresponds to Timed Rebeca semantics.
- Not depending on the system clock.

```
msgsrv calculate() {
  int i = 0;
  int s = ?(1,2,3,4,5);
  int x = i*s;
  inc = inc + 1;
  checkpoint(incremented,inc);
  self.calculate() after(1);
}
```

Experiment by iteratting the message server *calculate()*

- When executing for **65628,6** seconds the expected time is **65459** time units.
  Equals to 169,6 seconds or 0,3% of accumilated time-slides.
- Simulating for **60** seconds:
  Elapsed **131649** time units

Timed Rebeca Simulation Tool-set



Traceability of simulations.

Events

Message sending time
Message arrival time
Message Expiration
Rebec Name
Message Server Name
Parameters
Sender

Checkpoints

Time of checkpoint
Rebec name
Checkpoint label
Checkpoints terms

Statistical Evaluation Methods

- Main purpose to reason about some quantity connected to the model.
- The quantity is obtained using one observation as:
  - one checkpoint with a term value of interest.
  - two checkpoints with the elapsed time between them as a value of interest.
- Metrics are based on multiple observations.
  - Expected values (means).
  - Variance and standard deviations.
  - Worst- and best-case estimates.
- Simulations measurements from the same replicated behavior are combined to get an improved estimate of the real population.

Visualization Methods

- Large simulated data can offer difficulties to visualize.
- Provide raw and smoothed reduction plots.
  - For linear data smoothing we use moving averages with a given degree (N).
  - Reduce the data by picking periodic values from smoothed data.
- By using linear data smoothing and then reducing the data-set we are limiting showing wrong trends.

$$MA_M = \frac{1}{N}(i_M + i_{M-1} + \ldots + i_{M-(N-1)}) \tag{1}$$

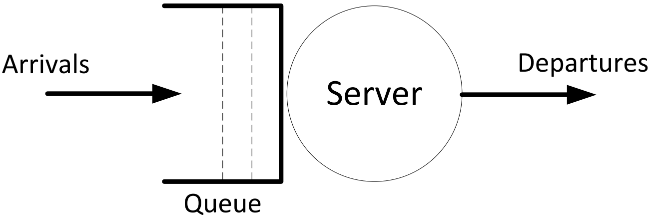
Performance Evaluation

In this project we name the performance evaluation methods:

- Response time analysis.
  - Measuring time between **two checkpoints**.
- Quantitative analysis.
  - Measuring a value of **checkpoints term**.
- Frequency analysis.
  - Using **events**.

Response Time Analysis Example.

Single server queue:



Response Time Analysis Example.

A Timed Rebec model of the single queue was simulated with *timedrebanalysis* 5 times, each with a timeout of 200 seconds. Response indicators were the following:

- Throughput of the queuing system.
- Queue waiting times.
- Server performance.

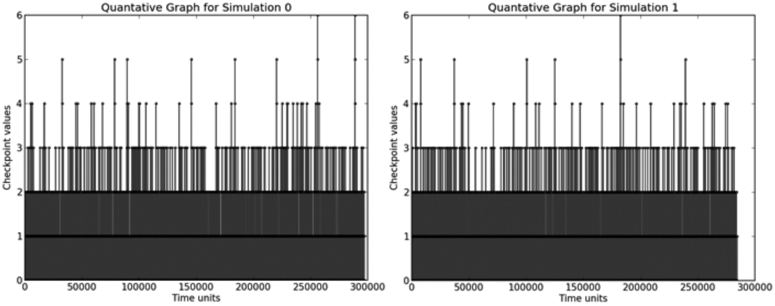
Response Indicator	Expected Response	Sample Standard Deviation	Minimum Value	Maximum Value	Median Value	Checkpoint Pairs
Throughput	10.3	3.7	2	42	11	258084
Queue waiting time	0.6	1.7	0	24	0	258084
Server Performance	3	1.4	1	5	3	258084

Quantitative Analysis Example.

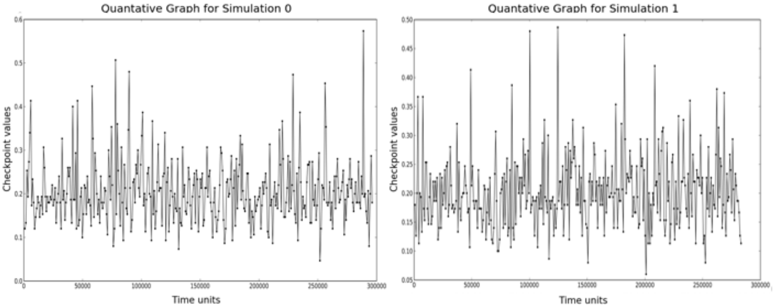
- How the size of the queue evolves over time.
- The value of the term is *queue.size()* at each time when a checkpoint occurs.

Simulation Number	Expected Value	Sample Standard Deviation	Minimum Value	Maximum Value	Median Value	Occurrences
1	0.2059	0.5254	0	6	0	53888
2	0.2037	0.5192	0	6	0	51642
3	0.2026	0.5112	0	5	0	50744
4	0.2037	0.5157	0	6	0	52781
5	0.2089	0.5263	0	7	0	49029
Total	0.2049	0.5196	0	7	0	258084

Single Queue: Visualization of the queue size for simulation 1 (Raw data).

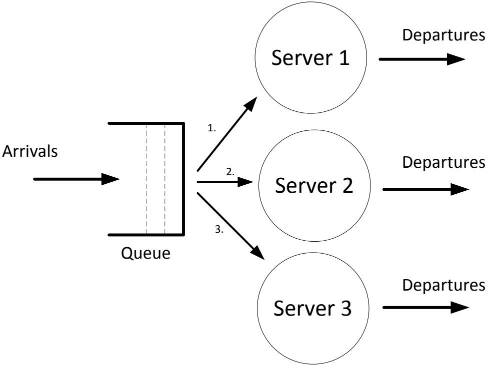


Single Queue: Visualization of the queue size for simulation 1 (Smoothed reduction with degree 150 and period 150).



Frequency Analysis Example.

Multi server queue:

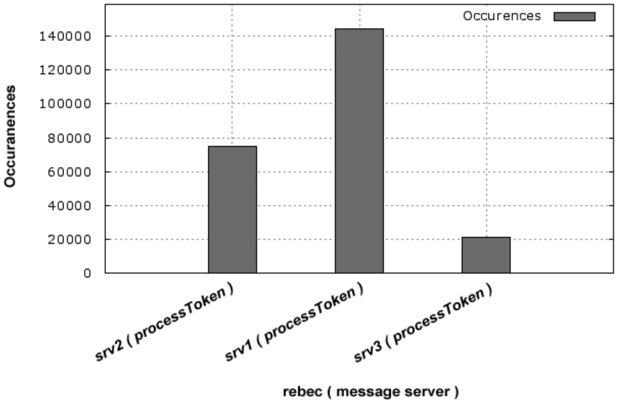


Frequency Analysis Example.

- A Timed Rebeca model of the multi-queuing system was simulated with *timedrebanalysis* 5 times, each with a timeout of 200 seconds.
- Interest in seeing the distribution among servers.
- Results obtained were:
  - Server 1, served 144391 requests (60%).
  - Server 2, served 74826 requests (31%).
  - Server 3, served 21344 requests (9%).

Frequency Analysis Example.

Visualization:

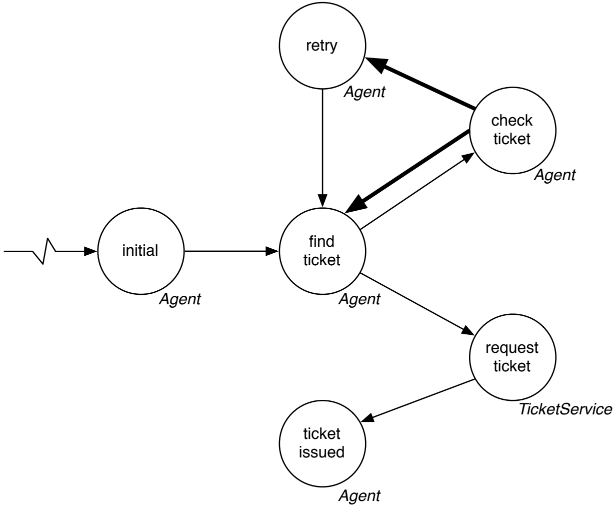


Ticket Service

Theses provides a formerly analyzed Ticket Service case study which we:

- Confirmed with verification.
  - Verification took between 1 and 2 seconds.
  - States explored were 40 to 5300.
- Verified additional properties.
- Provided more insights into the model with our performance analysis methods.

Ticket Service



Ticket Service

- Formerly analyzed in (Aceto et al., 2011).
- Used simulation (execution manner).

Setting	Request deadline	Check issued period	Retry request period	New request period	Service time 1	Service time 2	Result
1,2	2	1	1	1	3,4	7	Not issued
3	2	2	1	1	4	7	Not issued
4	2	2	1	1	3	7	**Ticket issued**

- Only got a issued ticket in setting 4.

Ticket Service

- We confirmed the results by writing a correctness property that violated when a issued ticket occured.

Setting	Request deadline	Check issued period	Retry request period	New request period	Service time 1	Service time 2	Max Ticket Requests	Result
1	2	1	1	1	3	7	7	Satisfied (170737 states)
2	2	1	1	1	4	7	7	Satisfied (199709 states)
3	2	2	1	1	4	7	7	Satisfied (153377 states)
4	2	2	1	1	3	7	7	**Violation** (6248 states)
5	2	2	1	1	2	7	7	**Violation** (4398 states)
6	2	3	1	1	2	7	7	**Violation** (4311 states)
7	2	4	1	1	2	7	7	**Violation** (4311 states)

- Issued tickets were possible in settings 4,5,6, and 7.

Ticket Service

- We checked if there is a setting where all tickets are issued.
- The property violated if a ticket was ever not issued.

Setting	Request deadline	Check issued period	Retry request period	New request period	Service time 1	Service time 2	Max Ticket Requests	Result
1	2	1	1	1	3	7	7	Violation (4584 states)
2	2	1	1	1	4	7	7	Violation (4926 states)
3	2	2	1	1	4	7	7	Violation (4440 states)
4	2	2	1	1	3	7	7	Violation (4360 states)
5	2	2	1	1	2	7	7	Violation (5227 states)
6	2	5	1	1	3	4	7	**Satisfied** (4807 states)
7	2	5	1	1	5	4	7	Violation (4807 states)
8	2	6	1	1	5	4	7	**Satisfied** (4807 states)

- Settings 6 and 8 had all tickets issued.

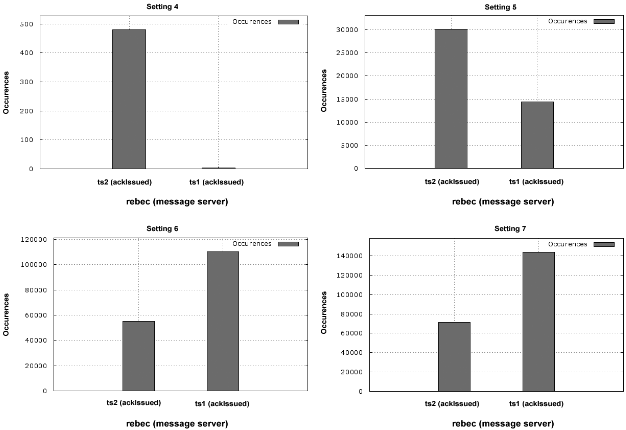
Ticket Service: Performance evaluation

- We wanted to performance evaluate settings 4,5,6, and 7 as all of them had a possible issued ticket.
- Setting 7 was most promising in regards to performance as all tickets got issued.

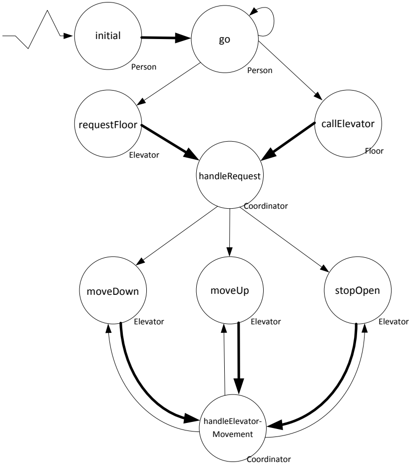
Setting	Expected response	Standard Deviation	Minimum response	Maximum response	Median response	Starting checkpoints	Checkpoint pairs
4	3	0	3	3	3	519350	614
5	2.1	0.1	2	3	2	511709	51476
6	4	0	4	4	4	363891	81585
7	3	0	3	3	3	573551	286948

- As promised, setting 7 had the most checkpoint pairs.
    - Interestingly, it had not the lowest response time.
- Interesting property of the model is that we get more issued tickets from settings 7, though we are setting the service time higher than settings 1 to 6.

Ticket Service: Frequency analysis for the ticket services



Elevator System



Elevator System

- Prototypical example for real-time analysis.
- Verification of correctness properties.
- Validation to examine and estimate performance of 4 implementations.
- Implementation are a combination of two policies.
  - Movement policy of an elevator.
  - Scheduling policy between the elevators.

Implementation #	Scheduling Policy	Movement Policy
1	Shortest distance	Up priority
2	Shortest distance	Maintain movement
3	Shortest distance with movement priority	Maintain movement
4	Shortest distance with load balancing	Maintain movement

Elevator System: Custom functions

- Used custom functions for operations that was not possible to do with Timed Rebeca syntax.
    - Checking if an integer was already in a list (Ignore floor request that are already in a elevator list)
    - Getting the absolute number for an integer (Calculate distance from the request and the current location of a elevator)
    - Function that check if there is a higher/lower integer in a list based on an two integers representing movement and location.

Elevator System: Verification

- Verified properties on a abstract version of the model (3 floors and 2 elevators).
  - Elevators only travel to and from the defined floors in the model.
  - Elevators only stop on defined floors in the model.
  - Elevators only stop on a floor if a request matching the floor is in the queue.
  - Request lists of the elevators never got size over 3.

```
monitorType() -> safety.

init(_) -> {ok, satisfied}.

stateChange(_,satisfied,Stack) ->
Actions = actions(Stack),
checkTermMinValue(Actions,elevatorLocation,0),
checkTermMaxValue(Actions,elevatorLocation,3),
checkTermValue(Actions,elevator1StopReqInList,-1),
checkTermValue(Actions,elevator2StopReqInList,-1),
checkTermMaxValue(elevator1QueueSize,3),
checkTermMaxValue(elevator2QueueSize,3).
```

Elevator System: Verification

- Specification satisfied for all implementations.
- Verification process took on average 110 seconds for each property.

Description	Movement Delays	Open Close Delays (1,2,3,4)	Result / States	Average Time
Location 0 $>$	2	1,2,4,6	Satisfied ≈40900 states	≈110 seconds
Location < 3	2	1,2,4,6	Satisfied ≈40900 states	≈110 seconds
Stop Queue 1	2	1,2,4,6	Satisfied ≈40900 states	≈110 seconds
Stop Queue 2	2	1,2,4,6	Satisfied ≈40900 states	≈110 seconds
Queue 1 < 3	2	1,2,4,6	Satisfied ≈40900 states	≈110 seconds
Queue 2 < 3	2	1,2,4,6	Satisfied ≈40900 states	≈110 seconds

Elevator System: Implementation 1 analysis.

- Implementation 1:
  - Scheduling policy: Shortest distance.
  - Movement policy: Up Priority.
- A simulation was carried out with *timedrebsim*
  - Issued 10 simulation.
  - Each simulated 1500 random requests with 2 time units delay.

Floor	Expected Response	Standard Deviation	Median Response	Max (WCET) Response	Min (BET) Response	Checkpoint pairs
1	58.5	76.2	29.0	683	1	4772
2	44.4	61.0	18.0	564	1	5591
3	33.1	46.1	14.0	467	1	6568
4	24.5	30.6	12.0	317	1	7361
5	20.6	21.6	13.0	196	1	7880
6	17.5	14.6	13.0	131	1	8182
7	14.6	10.9	12.0	85	1	8615
8	13.4	10.6	11.0	82	1	8966
9	14.7	12.3	11.0	89	1	8777
10	18.0	13.3	15.0	99	1	8442

Elevator System

- Implementation 2:
  - Scheduling policy: Shortest distance.
  - Movement policy: **Maintain movement**.
- A simulation was carried out with *timedrebsim*
  - Issued 10 simulation.
  - Each simulated 1500 random requests with 2 time units delay.

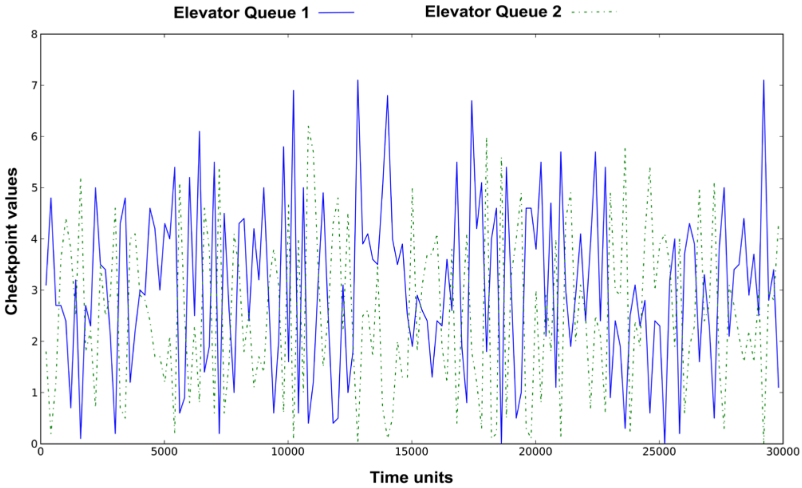
Floor	Expected Response	Standard Deviation	Median Response	Max (WCET) Response	Min (BET) Response	Checkpoint pairs
1	21.6	15.5	18.0	95	1	9004
2	17.3	14.1	12.0	87	1	9508
3	14.8	11.7	11.0	68	1	9926
4	14.6	10.5	12.0	72	1	9915
5	14.7	9.5	12.0	65	1	9762
6	14.6	9.7	12.0	62	1	9915
7	14.3	10.4	11.0	77	1	9919
8	14.8	11.8	11.0	80	1	9930
9	17.1	13.9	12.0	81	1	9555
10	21.7	15.5	17.0	86	1	9021

Elevator System

- Implementation 3:
  - Scheduling policy: **Shortest distance with movement priority**.
  - Movement policy: Maintain movement.
- A simulation was carried out with *timedrebsim*
  - Issued 10 simulation.
  - Each simulated 1500 random requests with 2 time units delay.

Floor	Expected Response	Standard Deviation	Median Response	Max (WCET) Response	Min (BET) Response	Checkpoint pairs
1	28.3	19.9	24.0	99	1	6767
2	22.4	17.9	17.0	92	1	7420
3	18.7	15.0	14.0	90	1	8168
4	16.7	12.5	14.0	78	1	8444
5	16.3	11.0	14.0	67	1	8457
6	16.2	10.9	14.0	63	1	8688
7	16.8	12.3	14.0	73	1	8449
8	18.6	15.0	14.0	79	1	8142
9	21.6	17.6	17.0	92	1	7691
10	28.1	19.9	24.0	103	1	6843

Elevator System

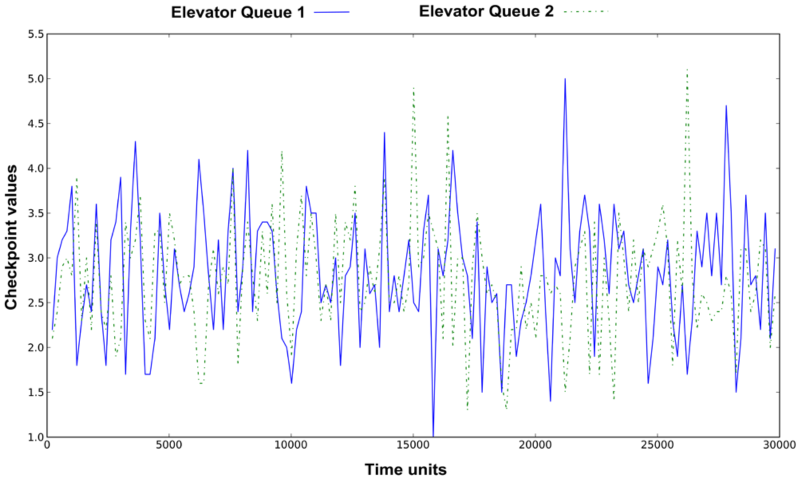


Elevator System

- Implementation 4:
  - Scheduling policy: **Shortest distance with load balancing**.
  - Movement policy: Maintain movement.
- A simulation was carried out with *timedrebsim*
  - Issued 10 simulation.
  - Each simulated 1500 random requests with 2 time units delay.

Floor	Expected Response	Standard Deviation	Median Response	Max (WCET) Response	Min (BET) Response	Checkpoint pairs
1	28.1	16.4	28.0	79	1	7096
2	22.9	15.3	21.0	76	1	7554
3	18.9	13.0	16.0	67	1	8161
4	16.8	10.9	14.0	64	1	8354
5	15.5	9.2	14.0	53	1	8600
6	15.6	9.5	14.0	52	1	8695
7	16.5	10.9	14.0	63	1	8457
8	19.2	13.2	16.0	66	1	8071
9	22.7	15.2	21.0	68	1	7627
10	28.4	16.7	28.0	85	1	7140

Elevator System



Elevator System: Conclusion

- Most inefficient implementation was the first one.
- Implementation 2 had most finished requests and the lowest expected response time.
- Implementation 4 had the lowest worst-case response estimate.
  - Interestingly, a lower expected response time was not achieved by implementing load balanced request dispatching.
- Interesting finding was that implementation 3 did not yield better results than implementation 2 and 4.

Implementation	**Expected response (Average)**	**Median response (Average)**	**Max response (Average)**	**Total finished requests**
1	25.93	14.8	271.3	75154
2	16.55	12.8	77.3	96455
3	20.37	16.6	83.6	79069
4	20.46	18.6	67.3	79755

Conclusion

- Extended Timed Rebeca to be able to model behaviors that was not possible before.
- Implemented a mapping that corresponded to Timed Rebeca's semantics.
- Verified formerly analyzed models.
- Introduced ways to get more insights into dynamic behaviors of a model.
- Thesis has detailed examples and case studies.
- Provides possibilities for future work.

Conclusion

Thank you!