Event-Based Analysis of Timed Rebeca Models Using SQL

Brynjar Magnusson, Ehsan Khamespanah, Marjan Sirjani and Ramtin Khosravi
Reykjavik University in Iceland and University of Tehran in Iran
AGERE@SPLASH
October 2014
Modeling and Verification of Timed Actors

- Realtime aspects of actors are important
- Schedulability of timed actors are important
- Model checking against realtime properties is required
- Reasoning about events is more interesting than state variables of actors
A Simple Timed Actor Model

• A customer wants to buy a ticket
• Issuing a ticket is a time consuming event
• There is network delay
**Timed Rebeca Model**

```rebeca
reactiveclass TicketService {
    knownrebecs {
        Agent a;
    }
    statevars {
        int issueDelay;
    }
    msgsrv requestTicket() {
        int issueDelay =?(3,4);
        delay(issueDelay);
        a.ticketIssued(1) after(2);
    }
}

reactiveclass Agent {
    knownrebecs {
        TicketService ts;
        Customer c;
    }
    msgsrv requestTicket() {
        ts.requestTicket() deadline(5) after(2);
    }
}

reactiveclass Customer {
    knownrebecs {
        Agent a;
    }
    msgsrv initial() {
        self.try();
    }
    msgsrv try() {
        a.requestTicket() after(2);
    }
    msgsrv ticketIssued(byte id) {
        c.ticketIssued(id) after(2);
    }
}

main {
    Agent a(ts, c):();
    TicketService ts(a):();
    Customer c(a):();
}
```
Timed Rebeca Model

```rebeca
reactiveclass TicketService {
    knownrebecs {
        Agent a;
    }
    statevars {
        int issueDelay;
    }
    msgsrv requestTicket() {
        int issueDelay =?(3,4);
        delay(issueDelay);
        a.ticketIssued(1) after(2);
    }
}
reactiveclass Agent {
    knownrebecs {
        TicketService ts;
        Customer c;
    }
    msgsrv requestTicket() {
        ts.requestTicket() deadline(5) after(2);
    }
}
reactiveclass Customer {
    knownrebecs {
        Agent a;
    }
    msgsrv initial() {
        self.try();
    }
    msgsrv try() {
        a.requestTicket() after(2);
    }
    msgsrv ticketIssued(byte id) {
        c.ticketIssued(id) after(2);
    }
}
main {
    Agent a(ts, c):();
    TicketService ts(a):();
    Customer c(a):();
}
```

Computation Time: 4
**Timed Rebeca Model**

```rebeca
reactiveclass TicketService {
    knownrebecs {
    }
    statevars {
        int issueDelay;
    }
    msgsrv requestTicket() {
        int issueDelay =?(3,4);
        delay(issueDelay);
        a.ticketIssued(1) after(2);
    }
}
reactiveclass Agent {
    knownrebecs {
        TicketService ts;
        Customer c;
    }
    msgsrv requestTicket() {
        ts.requestTicket() deadline(5) after(2);
    }
}
reactiveclass Customer {
    knownrebecs {
        Agent a;
    }
    msgsrv initial() {
        self.try();
    }
    msgsrv try() {
        a.requestTicket() after(2);
        self.try() after(30);
    }
    msgsrv ticketIssued(byte id) {
        c.ticketIssued(id) after(2);
    }
}
main {
    Agent a(ts, c):();
    TicketService ts(a):();
    Customer c(a):();
}
```
Timed Rebeca Model

reactiveclass TicketService {
    knownrebecs {
        Agent a;
    }
    statevars {
        int issueDelay;
    }
    msgsrv requestTicket() {
        int issueDelay = ?(3,4);
        delay(issueDelay);
        a.ticketIssued(1) after(2);
    }
}
reactiveclass Agent {
    knownrebecs {
        TicketService ts;
        Customer c;
    }
    msgsrv requestTicket() {
        ts.requestTicket() deadline(5) after(2);
    }
}
reactiveclass Customer {
    knownrebecs {
        Agent a;
    }
    msgsrv initial() {
        self.try();
    }
    msgsrv try() {
        a.requestTicket() after(2);
    }
    msgsrv ticketIssued(byte id) {
        c.ticketIssued(id) after(2);
    }
}
main {
    Agent a(ts, c):();
    TicketService ts(a):();
    Customer c(a):();
}
Timed Rebeca Model

```plaintext
reactiveclass TicketService {
    knownrebecs {
        Agent a;
    }
    statevars {
        int issueDelay;
    }
}

msgsrv requestTicket() {
    int issueDelay =?(3,4);
    delay(issueDelay);
    a.ticketIssued(1) after(2);
}

reactiveclass Agent {
    knownrebecs {
        TicketService ts;
        Customer c;
    }
    msgsrv requestTicket() {
        ts.requestTicket() deadline(5) after(2);
    }
}

reactiveclass Customer {
    knownrebecs {
        Agent a;
    }
    msgsrv initial() {
        self.try();
    }
    msgsrv try() {
        a.requestTicket() after(2);
    }
    msgsrv ticketIssued(byte id) {
        self.try() after(30);
    }
}

main {
    Agent a(ts, c):();
    TicketService ts(a):();
    Customer c(a):();
}
```
Analysis Support

• Floating Time Transition System for Schedulability and Deadlock-Freedom analysis

• Transforming to Erlang to simulate
Applications of Timed Rebeca

- Verification of Network-on-Chip (NoC) systems (model checking) - UT, Siamak Mohamadi
- Verification of Hadoop based systems (RU, Master Thesis) - UIUC, Indi Gupta
- Verification of Structural Health Monitoring system (model checking) - UIUC, Gul Agha
Analysis Support

- Floating Time Transition System for Schedulability and Deadlock-Freedom analysis
- Transforming to Erlang to simulate
- Event-Based property analysis using Timed Rebeca Simulation Engine
Event-Based Property Language for Timed Rebeca

- Computation takes palace by communication in actor models
- We need to take communication events into account
- A logic which events are its atomic propositions
- Easy to use by practitioners
Timed Event-Based Property Language (TeProp)

• A logic based on MTL (no branching operator)

• Influenced by property patterns of the specification of real-time systems and patterns in finite-state verification to address:

  • Maximum, minimum, exact, and bounded response to events

  • Periodic occurrence of events

  • Precedence relation between events
Introduction to TeProp

- Three temporal modalities
- Two operators
- Events as atomic propositions

Syntax of TeProp

\[
\phi ::= e | \neg \phi | \phi \land \phi | (\phi) | F_I e | F_I (e \leadsto \phi) | G_I (e \rightarrow \phi) | e B_I e
\]

\[
I ::= [\langle \text{Integer} \rangle, \langle \text{Integer} \rangle] | [\langle \text{Integer} \rangle, \text{end}]
\]

\[
e ::= \text{receiver}.\text{messageName}([\text{condition}])
\]
Intuitive Semantics

• Finally: $\textbf{F}[i_1, i_2] e$: An event matching $e$ will happen somewhere during the given interval.

• Before: $e_1 \textbf{B}[i_1, i_2] e_2$: With in the given interval, an event matching the first event happens at least once before an event matching the second one.

• Globally with implies: $\textbf{G}[i_1, i_2](e \rightarrow \phi)$: For all events matching $e$ during the given interval, the next formula must be satisfied.

• Finally with leads-to: $\textbf{F}[i_1, i_2](e \leadsto \phi)$: At least for one occurrence of an event matching $e$ in the given interval, the next formula holds true.
Property Patterns

- Maximum distance between an event and its reaction
  \[ G(e_1 \rightarrow F[0, x] \ e_2) \]

- Exact distance between an event and its reaction
  \[ G(e_1 \rightarrow F[x, x] \ e_2) \]

- Minimum distance between an event and its reaction
  \[ G(e_1 \rightarrow \neg F[0, x] \ e_2) \]
Property Patterns

• Periodic occurrence of events

\[ G(e \rightarrow (F[x, x] e \land \neg F[0, x - 1] e)) \land F[0, \infty] e \]

• Bounded response

\[ G(e_1 \rightarrow F[0, x] e_2) \]

• precedence relation between two events

\[ e_1 B[0, x] e_2 \]
Database Design and Mapping
TeProp to SQL Formula

- Occurrences of events are stored in database

- TeProp formulas transformed to SQL queries and SQL queries are executed over event traces

\[ e \rightarrow \text{exists}(e[0, 0]) \]

\[
\text{select alias}_e.\text{id from event}_e \text{ alias}_e \text{ where alias}_e.\text{id} > \text{alias}_\text{parent}.\text{id and alias}_e.\text{time between alias}_\text{parent}.\text{time} + i_1 \text{ and alias}_\text{parent}.\text{time} + i_2
\]

\[ \neg \phi \rightarrow \text{not(}\phi\text{)} \]

\[ \phi_1 \land \phi_2 \rightarrow (\phi_1) \text{ and } (\phi_2) \]

\[ F[i_1, i_2] e \rightarrow \text{exists}(e[i_1, i_2]) \]

\[ F[i_1, i_2] (e \leadsto \phi) \rightarrow \text{exists}(e[i_1, i_2]) \text{ and } (\phi) \]

\[ G[i_1, i_2] (e \rightarrow \phi) \rightarrow \text{not exists}((e[i_1, i_2]) \text{ and not}(\phi)) \]
Example of Mapping From TeProp Formulas to SQL formulas

\[
\text{SQL for } G (\text{senderAgent.start()} \rightarrow F[0, 10](\text{receiverAgent.send()}))
\]

```sql
select 'satisfied' from "base" t0_0 where (not exists(
    select t1_0.ID from "senderAgent_start" t1_0 where
    t1_0.ID > t0_0.ID and t1_0.time >= t0_0.time and
    not (exists(select t1_1.ID from "receiverAgent_send"
        t1_1 where t1_1.ID > t1_0.ID and t1_1.time between
        t1_0.time and t1_0.time + interval '10' second
    )))
)
```
Some Issues in Simulation of Timed Rebeca Models

• When to stop simulation

• It depends on the behaviors of systems

• Number of simulation traces

• Computing confidence interval based on the number of traces

\[ N = \frac{\gamma_2 \times \epsilon}{\hat{\mu}_z} \]

\[ \gamma_2 = 2(1 + \sqrt{\epsilon})(1 + 2\sqrt{\epsilon})(1 + \frac{\ln 3/2}{\ln 2/\delta})\gamma \]

\[ \gamma = \frac{4}{\epsilon^2}(\epsilon - 2)\ln(2/\delta) \]
Experimental Results

Control Systems

Toxic Level

Leave the Lab

Rescue Scientist
Experimental Results

- Sensor network example

<table>
<thead>
<tr>
<th>Property</th>
<th>Setting / Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>The scientist will not die:</td>
<td>0%   100%   0%   0%   0%   100%   100%</td>
</tr>
<tr>
<td>$\neg F[0,\text{end}] \text{admin.scientistDead()}$</td>
<td></td>
</tr>
<tr>
<td>The rescue team never went to rescue:</td>
<td>0%   0%   0%   0%   0%   0%   100%</td>
</tr>
<tr>
<td>$\neg F[0,\text{end}] \text{rescue.go()}$</td>
<td></td>
</tr>
<tr>
<td>Admin never misses an acknowledgment</td>
<td>0%   0%   0%   0%   0%   0%   100%</td>
</tr>
<tr>
<td>as result of ordering of events within a time</td>
<td></td>
</tr>
<tr>
<td>unit: $G(\text{admin.checkScientistAck()} \rightarrow \neg F[0,0] \text{admin.ack()})$</td>
<td></td>
</tr>
</tbody>
</table>

- Multi-flight booking

<table>
<thead>
<tr>
<th>Property</th>
<th>Setting / Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>The first ticket is successfully booked:</td>
<td>7%   52%   13%   28%   0%   90%   100%</td>
</tr>
<tr>
<td>$F[0,\text{end}] \text{customer.flightBooked}(f == \text{“1”} \land \text{successful == “true”})$</td>
<td></td>
</tr>
<tr>
<td>The second ticket is successfully booked:</td>
<td>9%   54%   50%   48%   0%   100%   100%</td>
</tr>
<tr>
<td>$F[0,\text{end}] \text{customer.flightBooked}(f == \text{“2”} \land \text{successful == “true”})$</td>
<td></td>
</tr>
<tr>
<td>All tickets are successfully booked:</td>
<td>2%   31%   7%   19%   0%   90%   100%</td>
</tr>
<tr>
<td>$\neg F[0,\text{end}] \text{customer.flightBooked(successful == “false”)}$</td>
<td></td>
</tr>
<tr>
<td>Booking occurred 3 or more time units before the reservation ran out:</td>
<td>0%   75%   0%   30%   0%   57%   100%</td>
</tr>
<tr>
<td>$F[0,\text{end}] (\text{ws1.bookFlight()} \sim F[3,\text{end}] \text{ws1.reservationExpired()}) \lor$</td>
<td></td>
</tr>
<tr>
<td>$F[0,\text{end}] (\text{ws2.bookFlight()} \sim F[3,\text{end}] \text{ws2.reservationExpired()})$</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

• A new event-based property language (TeProp)
• Using simulation traces for analysis of the model
• Using Database as the repository of simulation traces
• Mapping TeProp to SQL to make database able to analyse the stored simulation traces
Thank you