Event-Based Analysis of Timed Rebeca Models Using SQL

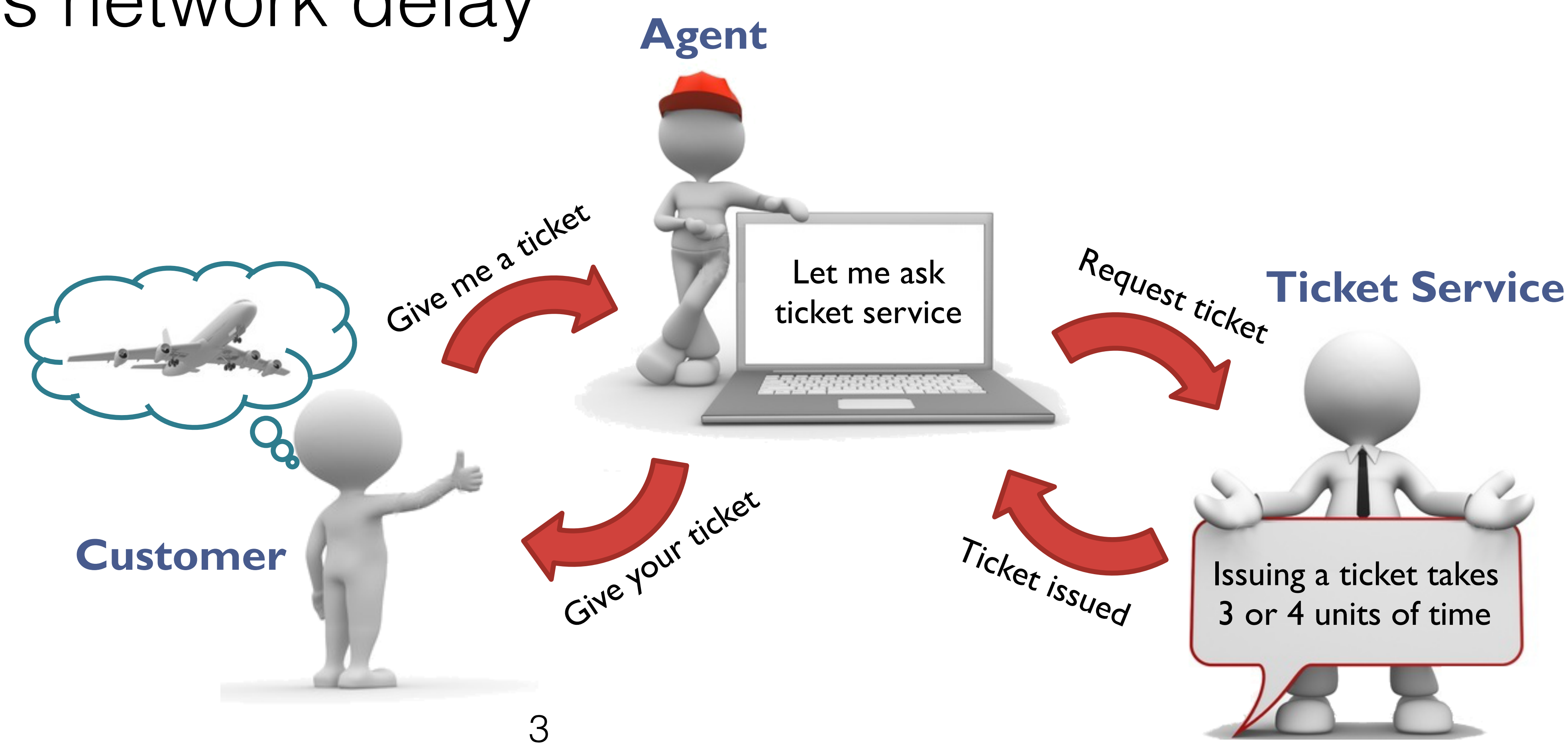
Brynjar Magnusson, Ehsan Khamespanah, Marjan Sirjani and Ramtin Khosravi
Reykjavik University in Iceland and University of Tehran in Iran
AGERE@SPLASH
October 2014

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

```
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);
    }
    msgsrv ticketIssued(byte id) {
        c.ticketIssued(id) 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):();
}
```

Timed Rebeca Model

Computation Time

```
reactiveclass TicketService {
    knownrebecs {

    }
    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);
    }
    msgsrv ticketIssued(byte id) {
        c.ticketIssued(id) 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):();
}
```

Timed Rebeca Model

Computation Time

Network Delay

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

     sgsrv ticketIssued(byte id) {
         self.try() after(30);
    }
}
main {
    Agent a(ts, c):();
    TicketService ts(a):();
    Customer c(a):();
}
```

Timed Rebeca Model

Computation Time

Network Delay

Periodic Event

```
reactiveclass TicketService {
    knownrebecs {
    }
    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);
    }
    msgsrv ticketIssued(byte id) {
        c.ticketIssued(id) 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):();
}
```

Timed Rebeca Model

```
reactiveclass TicketService {
    knownrebecs {

    }
    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);
    }
    msgsrv ticketIssued(byte id) {
        c.ticketIssued(id) 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):();
}
```

Computation Time

Network Delay

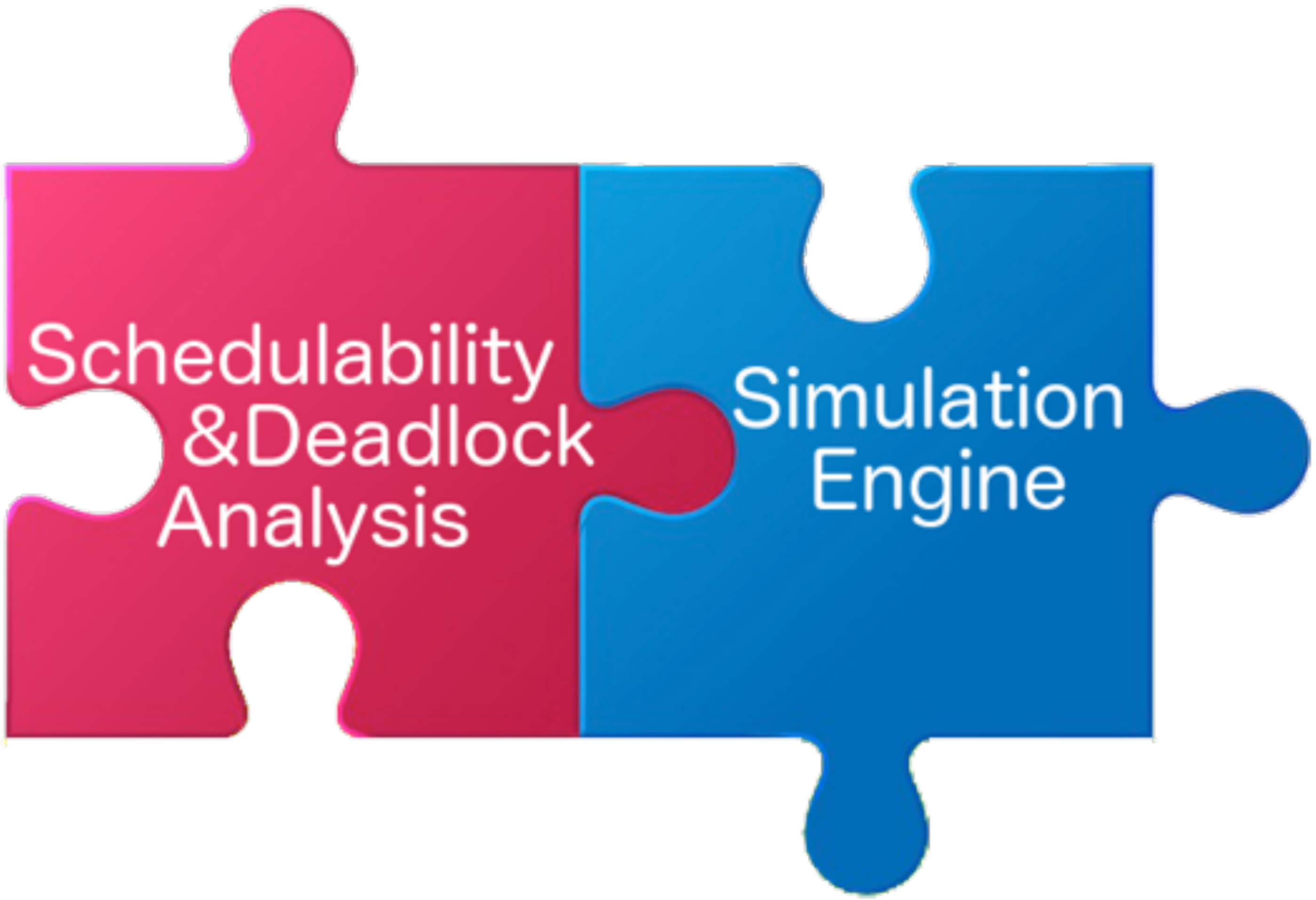
Periodic Event

Message Expiration Time

Analysis Support

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

- Transforming to Erlang to simulate



Applications of Timed Rebeca

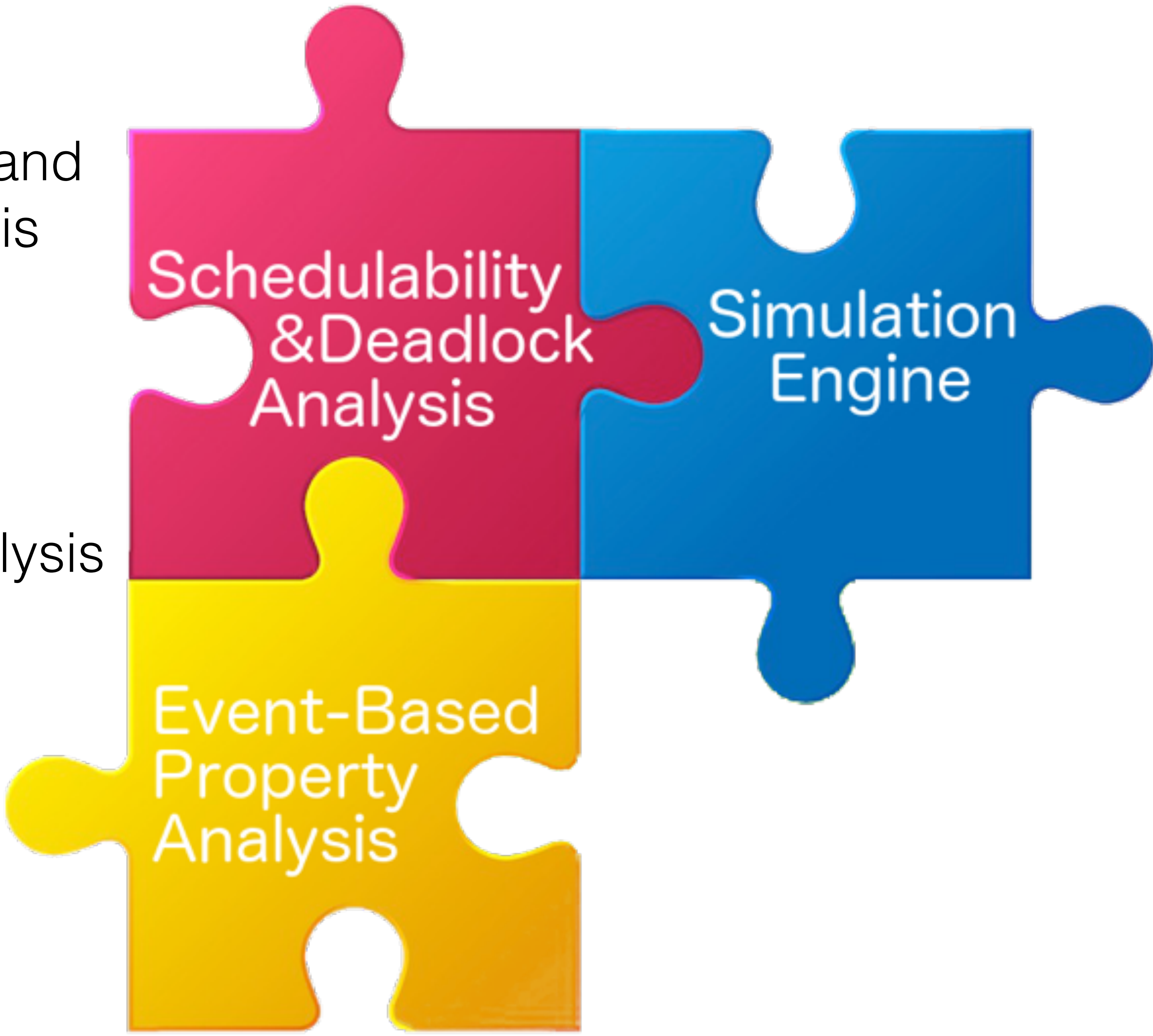
- Verification of Network-on-Chip (NoC) systems (model checking) - UT, Siamak Mohamadi

- 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

$$
\begin{aligned}
\phi &::= e \mid \neg\phi \mid \phi \wedge \phi \mid (\phi) \mid F_I\ e \mid F_I\ (e \rightsquigarrow \phi) \mid G_I\ (e \rightarrow \phi) \mid e\ B_I\ e \\
I &::= [\langle Integer \rangle,\ \langle Integer \rangle] \mid [\langle Integer \rangle,\ end] \\
e &::= receiver.messageName([condition])
\end{aligned}
$$

Intuitive Semantics

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

- Before: $e_1\ \ \mathbf{B}[i_1, i_2]\ \ e_2$: 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: $\mathbf{G}[i_1, i_2](e \rightarrow \phi)$: For all events matching e during the given interval, the next formula myst be satisfied.

- Finally with leads-to: $\mathbf{F}[i_1, i_2](e \rightsquigarrow \phi)$: 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

$$\mathbf{G}(e_1 \rightarrow \mathbf{F}[0, x]\; e_2)$$

- Exact distance between an event and its reaction

$$\mathbf{G}(e_1 \rightarrow \mathbf{F}[x, x]\; e_2)$$

- Minimum distance between an event and its reaction

$$\mathbf{G}(e_1 \rightarrow \neg\mathbf{F}[0, x]\; e_2)$$

Property Patterns

- Periodic occurrence of events

$$\mathbf{G}(e \rightarrow (\mathbf{F}[x,x]\, e \wedge \neg\mathbf{F}[0,x-1]\, e)) \wedge \mathbf{F}[0,\infty]\, e$$

- Bounded response

$$\mathbf{G}(e_1 \rightarrow \mathbf{F}[0,x]\, e_2)$$

- precedence relation between two events

$$e_1\,\mathbf{B}[0,x]\, e_2$$

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

$$e \rightarrow \textbf{exists}(\texttt{e}[\texttt{0},\texttt{0}])$$

$$\textbf{select}\ \texttt{alias}_{\texttt{e}}\texttt{.id}\ \textbf{from}\ \texttt{event}_{\texttt{e}}\ \texttt{alias}_{\texttt{e}}\ \textbf{where}\ \texttt{alias}_{\texttt{e}}\texttt{.id} > \texttt{alias}_{\texttt{parent}}\texttt{.id}\ \textbf{and}\ \texttt{alias}_{\texttt{e}}\texttt{.time}\ \textbf{between}\ \texttt{alias}_{\texttt{parent}}\texttt{.time} + \texttt{i}_1\ \textbf{and}\ \texttt{alias}_{\texttt{parent}}\texttt{.time} + \texttt{i}_2$$

$$\begin{array}{ll}
\neg\phi & \rightarrow \textbf{not}(\phi) \\
\phi_1 \wedge \phi_2 & \rightarrow (\phi_1)\ \textbf{and}\ (\phi_2) \\
\mathbf{F}[i_1, i_2]\, e & \rightarrow \textbf{exists}(\texttt{e}[\texttt{i}_1, \texttt{i}_2]) \\
\mathbf{F}[i_1, i_2]\, (e \rightsquigarrow \phi) & \rightarrow \textbf{exists}(\texttt{e}[\texttt{i}_1, \texttt{i}_2])\ \textbf{and}\ (\phi) \\
\mathbf{G}[i_1, i_2]\, (e \rightarrow \phi) & \rightarrow \textbf{not exists}((\texttt{e}[\texttt{i}_1, \texttt{i}_2])\ \textbf{and}\ \textbf{not}(\phi))
\end{array}$$

Example of Mapping From TeProp Formulas to SQL formulas

SQL for G (senderAgent.start() → F[0, 10](receiverAgent.send()))

```
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

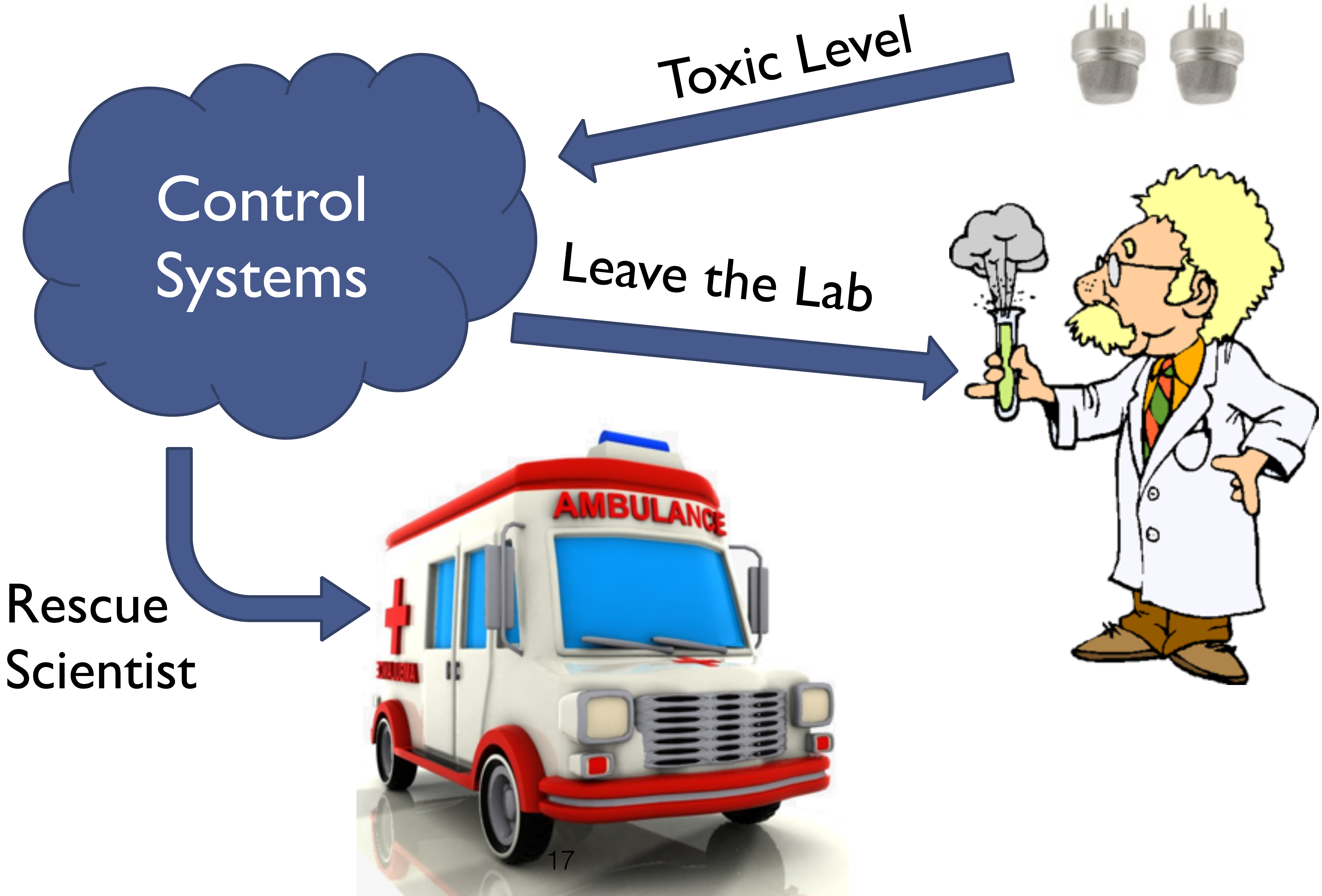
- When to stop simulation
  - It depends on the behaviors of systems
- Number of simulation traces
  - Computing confidence interval based on the number of traces

$$N = \frac{\Upsilon_2 \times \epsilon}{\hat{\mu}_z}$$

$$\Upsilon_2 = 2(1 + \sqrt{\epsilon})(1 + 2\sqrt{\epsilon})(1 + \frac{ln3/2}{ln2/\delta})\Upsilon$$

$$\Upsilon = \frac{4}{\epsilon^2}(e-2)ln(2/\delta)$$

Experimental Results



Experimental Results

- Sensor network example

Property	Setting / Result 1	2	3	4	5	6	7
The scientist will not die: $\neg\mathbf{F}\ [0, end]$ admin.scientistDead()	0%	100%	0%	0%	0%	100%	100%
The rescue team never went to rescue: $\neg\mathbf{F}\ [0, end]$ rescue.go()	0%	0%	0%	0%	0%	0%	100%
Admin never misses an acknowledgment as result of ordering of events within a time unit: $\mathbf{G}$(admin.checkScientistAck() $\rightarrow \neg\mathbf{F}\ [0,0]$ admin.ack())	0%	0%	0%	0%	0%	0%	100%

- Multi-flight booking

Property	Setting / Result 1	2	3	4	5	6	7
The first ticket is successfully booked: $\mathbf{F}\ [0, end]$ customer.flightBooked(f == "1" $\wedge$ successful == "true")	7%	52%	13%	28%	0%	90%	100%
The second ticket is successfully booked: $\mathbf{F}\ [0, end]$ customer.flightBooked(f == "2" $\wedge$ successful == "true")	9%	54%	50%	48%	0%	100%	100%
All tickets are successfully booked: $\neg\mathbf{F}\ [0, end]$ customer.flightBooked(successful == "false")	2%	31%	7%	19%	0%	90%	100%
Booking occurred 3 or more time units before the reservation ran out: $\mathbf{F}\ [0, end]$ (ws1.bookFlight() $\rightsquigarrow \mathbf{F}\ [3, end]$ ws1.reservationExpired()) $\vee$ $\mathbf{F}\ [0, end]$ (ws2.bookFlight() $\rightsquigarrow \mathbf{F}\ [3, end]$ ws2.reservationExpired())	0%	75%	0%	30%	0%	57%	100%

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