Analysis of Timing Constraints in Heterogeneous Middleware Interactions

Ajay Kattepur[1], Nikolaos Georgantas[2], Georgios Bouloukakis[2] & Valérie Issarny[2]

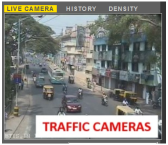
[1]PERC, TCS Innovation Labs, Mumbai, India
[2]MiMove Team, Inria Paris-Rocquencourt, France

ICSOC, Goa, November 2015

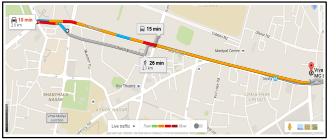
Motivation

What is the current traffic congestion on Mahatma Gandhi Road ?

Crowd sourced Information



Traffic Cameras



Google “real time” traffic

post **post** **post**

Storage with limited validity “latest report” **(lease)**

get **get** **get**

• • •

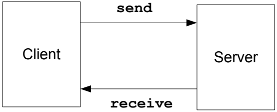
Mobile Subscribers Connect/Disconnect **(timeout)**

Motivation

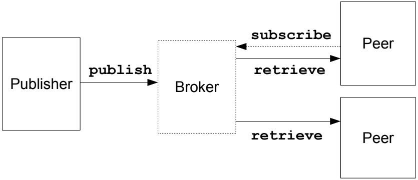
- Future Internet Application Middleware:
  - Web Services – *Client-Service* (CS)
  - Data Feeds, IoT – *Publish-Subscribe* (PS)
  - Crowd-sourcing – *Tuple Spaces* (TS)
- *eXtensible Service Bus* (XSB) Middleware – unifying connector for CS|PS|TS
- Timing Analysis of Interactions:
  - Data validity constraint with `lease` parameter
  - Intermittent subscriber availability with `timeout` parameter
- System designers can tune timing parameters for *Transaction Success* and *Latency*

Outline

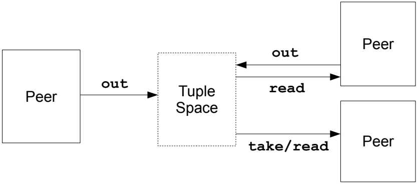
Middleware Interaction Paradigms



- Client–Service (CS)
  - Tight Time Coupling
  - Tight Space Coupling



- Publish–Subscribe (PS)
  - Time Decoupling
  - Space Decoupling



- Tuple Space (TS)
  - Time Decoupling
  - Space Decoupling

XSB Model

- *eXtensible Service Bus* (XSB) Middleware Connector[1]

Primitives	Arguments
post	mainscope, subscope, data, lease
get	↑mainscope, ↑subscope, ↑data, timeout

- Functional and non-functional (timed) behavior of CS|PS|TS interactions [2]

	Native Primitives	XSB Primitives
CS	send(destination, operation, message)	post(destination, operation, message, lease(0))
	receive(↑source, ↑operation, ↑message, timeout)	get(↑source, ↑operation, ↑message, timeout)
PS	publish(broker, filter, event, lease)	post(broker, filter, event, lease)
	retrieve(↑broker, ↑filter, ↑event, timeout)	get(↑broker, ↑filter, ↑event, timeout)
TS	out(tspace, template, tuple, lease)	post(tspace, template, tuple, lease)
	take(↑tspace, ↑template, ↑tuple, timeout)	get(↑tspace, ↑template, ↑tuple, timeout)
	read(↑tspace, ↑template, ↑tuple, timeout)	get(↑tspace, ↑template, ↑tuple, timeout)

[1] http://xsb.inria.fr/

[2] S. S. Lam, "Protocol Conversion", *IEEE Trans. on Software Engineering*, v. 14, n. 3, 1988.

XSB Model

- *eXtensible Service Bus* (XSB) Middleware Connector[1]

Primitives	Arguments
post	mainscope, subscope, data, lease
get	↑mainscope, ↑subscope, ↑data, timeout

- Functional and non-functional (timed) behavior of CS|PS|TS interactions [2]

	Native Primitives	XSB Primitives
CS	**send**(destination, operation, message)	**post**(destination, operation, message, lease(0))
	receive(↑source, ↑operation, ↑message, timeout)	**get**(↑source, ↑operation, ↑message, timeout)
PS	**publish**(broker, filter, event, lease)	**post**(broker, filter, event, lease)
	retrieve(↑broker, ↑filter, ↑event, timeout)	**get**(↑broker, ↑filter, ↑event, timeout)
TS	**out**(tspace, template, tuple, lease)	**post**(tspace, template, tuple, lease)
	take(↑tspace, ↑template, ↑tuple, timeout)	**get**(↑tspace, ↑template, ↑tuple, timeout)
	read(↑tspace, ↑template, ↑tuple, timeout)	**get**(↑tspace, ↑template, ↑tuple, timeout)

[1] http://xsb.inria.fr/

[2] S. S. Lam, "Protocol Conversion", *IEEE Trans. on Software Engineering*, v. 14, n. 3, 1988.

XSB Model

- Operations with active and inactive time interval constraints
  1. `lease`: max active interval for `post` (data validity)
  2. `timeout`: invariant active interval for `get` (subscriber availability)
  3. `lease` ≈ 0 for CS (tight time coupling)
- Models time-correlation between `post` and `get` operations for forming end-to-end XSB transactions (CS↔PS↔TS)[3]
- Data processing, transmission and queueing times assumed negligible compared to `lease`/`timeout` intervals
- This corresponds to a $G/G/\infty/\infty$ queueing model

[3] A. Kattepur, N. Georgantas & V. Issarny, "QoS Analysis in Heterogeneous Choreography Interactions", ICSOC, 2013.

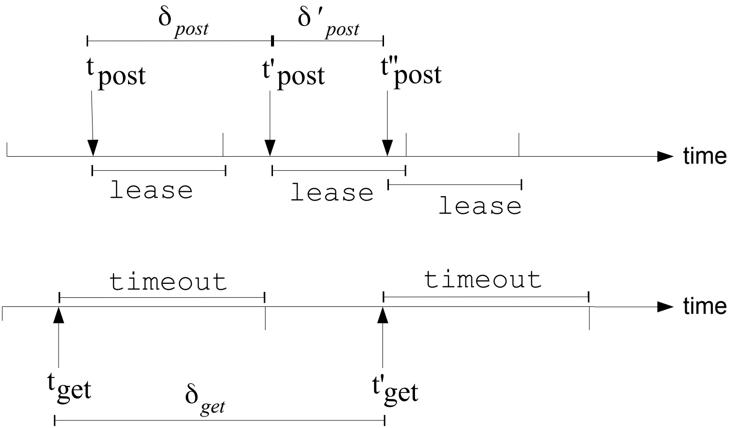
XSB Model

- Operations with active and inactive time interval constraints
  1. `lease`: max active interval for `post` (data validity)
  2. `timeout`: invariant active interval for `get` (subscriber availability)
  3. `lease` ≈ 0 for CS (tight time coupling)
- Models time-correlation between `post` and `get` operations for forming end-to-end XSB transactions (CS↔PS↔TS)[3]
- Data processing, transmission and queueing times assumed negligible compared to `lease`/`timeout` intervals
- This corresponds to a $G/G/\infty/\infty$ queueing model

[3] A. Kattepur, N. Georgantas & V. Issarny, "QoS Analysis in Heterogeneous Choreography Interactions", ICSOC, 2013.

Outline

XSB Timing Model



XSB Timing Model

- Disjunctive conditions for Transaction Success:
  1. If post occurs first (data posted):

$$t_{\texttt{post}} < t_{\texttt{get}} < t_{\texttt{post}} + \texttt{lease} \tag{1}$$

     - get occurs before lease – transaction *successful*, or
     - lease is reached – transaction is a *failure*
  2. If get occurs first (subscriber connected):

$$t_{\texttt{get}} < t_{\texttt{post}} < t_{\texttt{get}} + \texttt{timeout} \tag{2}$$

     - post occurs before timeout – transaction *successful*, or
     - timeout is reached – get operation yields no transaction
- Represents individual CS|PS|TS interactions and heterogeneous interconnections between them

XSB Timing Model

- Disjunctive conditions for Transaction Success:
  1. If `post` occurs first (data posted):

$$t_{\texttt{post}} < t_{\texttt{get}} < t_{\texttt{post}} + \texttt{lease} \tag{1}$$

     - `get` occurs before `lease` – transaction *successful*, or
     - `lease` is reached – transaction is a *failure*

  2. If `get` occurs first (subscriber connected):

$$t_{\texttt{get}} < t_{\texttt{post}} < t_{\texttt{get}} + \texttt{timeout} \tag{2}$$

     - `post` occurs before `timeout` – transaction *successful*, or
     - `timeout` is reached – `get` operation yields no transaction

- Represents individual CS|PS|TS interactions and heterogeneous interconnections between them

XSB Timing Model

- Disjunctive conditions for Transaction Success:
  1. If `post` occurs first (data posted):

$$t_{\texttt{post}} < t_{\texttt{get}} < t_{\texttt{post}} + \texttt{lease} \tag{1}$$

     - `get` occurs before `lease` – transaction *successful*, or
     - `lease` is reached – transaction is a *failure*

  2. If `get` occurs first (subscriber connected):

$$t_{\texttt{get}} < t_{\texttt{post}} < t_{\texttt{get}} + \texttt{timeout} \tag{2}$$

     - `post` occurs before `timeout` – transaction *successful*, or
     - `timeout` is reached – `get` operation yields no transaction

- Represents individual CS|PS|TS interactions and heterogeneous interconnections between them

Outline

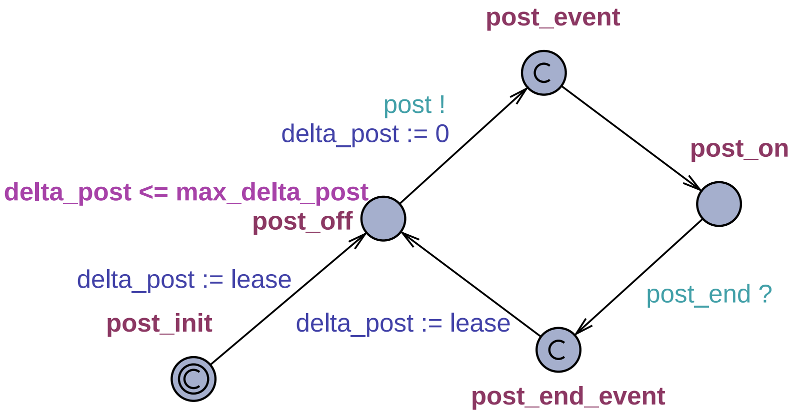
Timed Automata Model

- A timed automaton[4] is a finite automaton extended with real-valued clock variables.
- Clocks used to control `post` and `get` operations with active and inactive intervals
- Formal model on UPPAAL[5] allows verification
- Two role automata *Poster*/*Getter* interact via the *Glue* automaton
- `c!` (sending action) synchronizes with the transition of another automaton labeled with `c?` (receiving action)

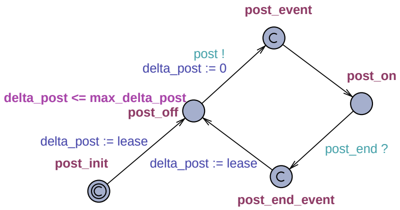
[4] R. Alur and D. L. Dill, "A Theory of Timed Automata", *Theoretical Computer Science*, 1994.

[5] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL4.0", Aalborg University, Denmark, 2006.

Poster - Timed Automata Model

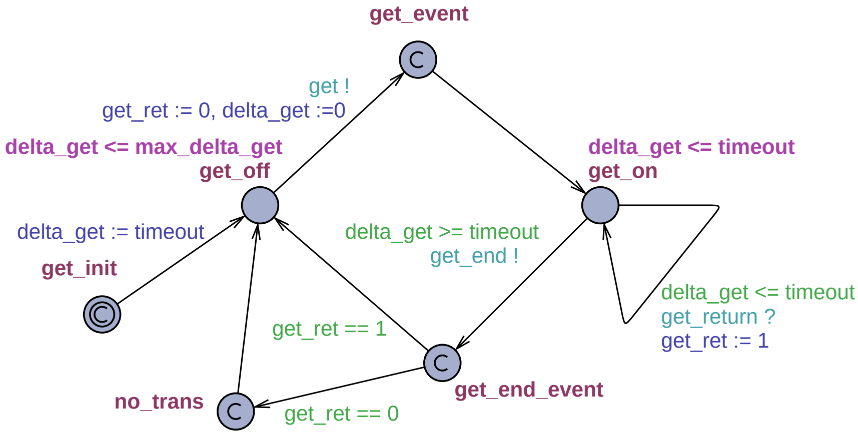


Poster - Timed Automata Model

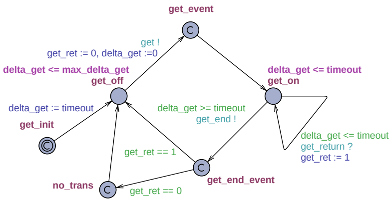


- Uniform distribution of inactive `post` intervals
- Disallows concurrent active posts via *Glue* feedback
- Arrival process for one of the infinite on-demand servers of $G/G/\infty/\infty$ model

Getter - Timed Automata Model

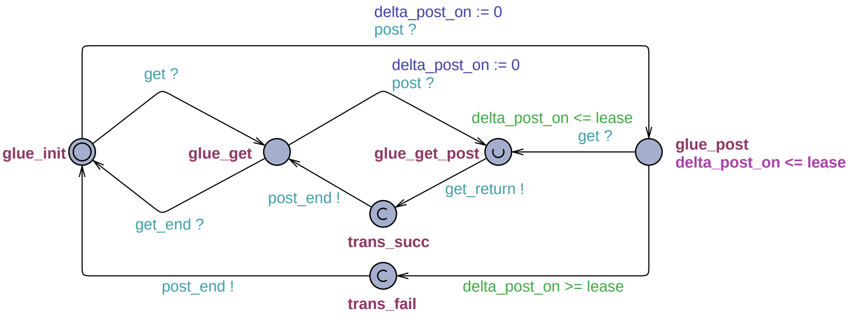


Getter - Timed Automata Model

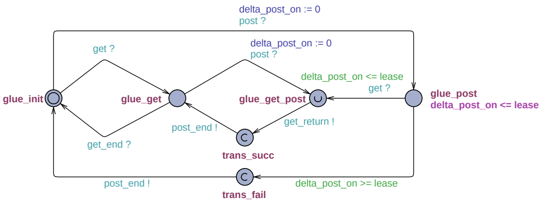


- Uniform distribution of inactive `get` intervals
- Controls active `get` intervals with a constant `timeout`
- Multiple posts may be received during an active interval
- Detects when no transaction is concluded (`no_trans` event)

Glue - Timed Automata Model



Glue - Timed Automata Model



- Controls max active `post` intervals with a constant `lease` lifetime
- Synchronizes between concurrent `post` and `get` operations
- Detects when synchronization is successful, with the `trans_succ` and `trans_fail` events

Verification

- Reachability (`E<>`φ) and Safety (`A[]`φ) properties verified
- Successful transaction (overlap in time between active post and active get)

```
A[] glue.trans_succ imply (poster.post_on and getter.get_on
                        and (delta_post==0 or delta_get==0))
```
(3)

- Failed transaction (ongoing inactive get interval entirely includes a terminating active post interval)

```
A[] glue.trans_fail imply (poster.post_on and getter.get_off
        and delta_post==lease and delta_get-timeout>=lease)
```
(4)

- No transaction concluded (ongoing inactive post interval entirely includes a terminating active get interval)

```
A[] getter.no_trans imply (getter.get_on and poster.post_off
       and delta_get==timeout and delta_post-lease>=timeout)
```
(5)

Verification

- Reachability (E<>φ) and Safety (A[]φ) properties verified
- Successful transaction (overlap in time between active post and active get)

```
A[] glue.trans_succ imply (poster.post_on and getter.get_on
                          and (delta_post==0 or delta_get==0))
```
(3)

- Failed transaction (ongoing inactive get interval entirely includes a terminating active post interval)

```
A[] glue.trans_fail imply (poster.post_on and getter.get_off
        and delta_post==lease and delta_get-timeout>=lease)
```
(4)

- No transaction concluded (ongoing inactive post interval entirely includes a terminating active get interval)

```
A[] getter.no_trans imply (getter.get_on and poster.post_off
        and delta_get==timeout and delta_post-lease>=timeout)
```
(5)

Verification

- Reachability (`E<>`φ) and Safety (`A[]`φ) properties verified
- Successful transaction (overlap in time between active post and active get)

```
A[] glue.trans_succ imply (poster.post_on and getter.get_on
                    and (delta_post==0 or delta_get==0))
```
(3)

- Failed transaction (ongoing inactive get interval entirely includes a terminating active post interval)

```
A[] glue.trans_fail imply (poster.post_on and getter.get_off
        and delta_post==lease and delta_get-timeout>=lease)
```
(4)

- No transaction concluded (ongoing inactive post interval entirely includes a terminating active get interval)

```
A[] getter.no_trans imply (getter.get_on and poster.post_off
       and delta_get==timeout and delta_post-lease>=timeout)
```
(5)

Verification

- Reachability (`E<>`φ) and Safety (`A[]`φ) properties verified
- Successful transaction (overlap in time between active post and active get)

```
A[] glue.trans_succ imply (poster.post_on and getter.get_on
                     and (delta_post==0 or delta_get==0))
```
(3)

- Failed transaction (ongoing inactive get interval entirely includes a terminating active post interval)

```
A[] glue.trans_fail imply (poster.post_on and getter.get_off
        and delta_post==lease and delta_get-timeout>=lease)
```
(4)

- No transaction concluded (ongoing inactive post interval entirely includes a terminating active get interval)

```
A[] getter.no_trans imply (getter.get_on and poster.post_off
      and delta_get==timeout and delta_post-lease>=timeout)
```
(5)

Verification

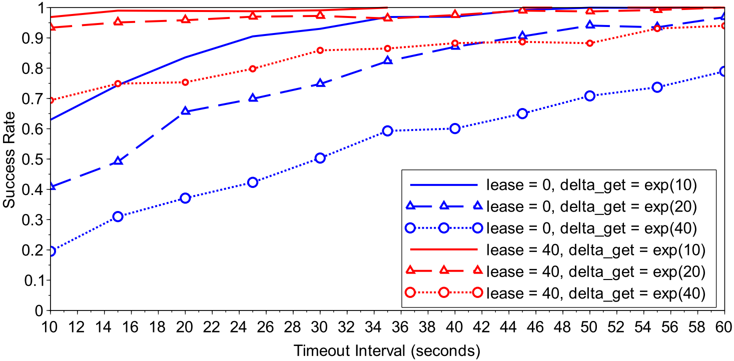
- Successful transactions – durations and relative positions of active/inactive `post` and `get`
- Dependence on deterministic parameters `lease`, `timeout` and stochastic intervals $\delta_{\texttt{post}}$ (on/off), $\delta_{\texttt{get}}$ (on/off)
- General formal conditions for successful XSB transactions – potentially tunable system parameters
- We perform experiments to quantify the effect of varying these parameters for successful transactions

Outline

Statistical Simulation: Transaction Success Rates

- Test the effect of varying `lease` and `timeout` periods on transaction success rates
- Exponential distributions for intervals between successive `post` operations ($\delta_{\texttt{post}}$) and successive `get` operations ($\delta_{\texttt{get}}$)
- No queueing effects included in the model
- This simulates an $M/G/\infty/\infty$ queueing model
- Simulation run for $10,000$ `get` operations to collect transaction statistics – transaction success conditions from formal analysis

Statistical Simulation: Transaction Success Rates

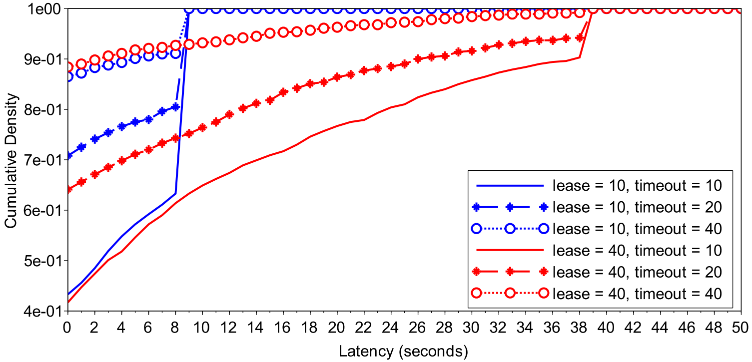


- Increasing `timeout` periods for individual `lease` values improves the success rate
- Success Rate is severely bounded by `lease` periods – evident in the CS case

Statistical Simulation: Latency vs. Success Rate

- Trade-off between end-to-end latency and transaction success rate
- Cumulative latency distributions for transactions
- All failed transactions are pegged to the max value `lease`
- Guidelines for system designers to set the `lease` and `timeout` periods for successful transactions with acceptable latency

Statistical Simulation: Latency vs. Success Rate



- Lower lease periods (e.g., CS case) produce markedly improved latency, however, with lower success rate
- With higher levels of lease periods (typically PS/TS), we notice high success rates, but also higher latency

Experimental Comparison: XSB Implementation

- Two middleware experimental setups:
  - `lease` $= 0$ transactions, experiment with the DPWS[6] CS middleware
  - `lease` > 0 transactions, experiment with the JMS[7] PS middleware
- Includes concurrent posts and queueing (queueing delays are negligible)
- This corresponds to an $M/G/1/\infty$ queueing model
- Each experiment run for at least 2 hours to ensure close statistical samples for $\delta_{\texttt{post}}$ and $\delta_{\texttt{get}}$ distributions

---

[6] http://ws4d.e-technik.uni-rostock.de/jmeds

[7] http://activemq.apache.org

Experimental Comparison: XSB Implementation

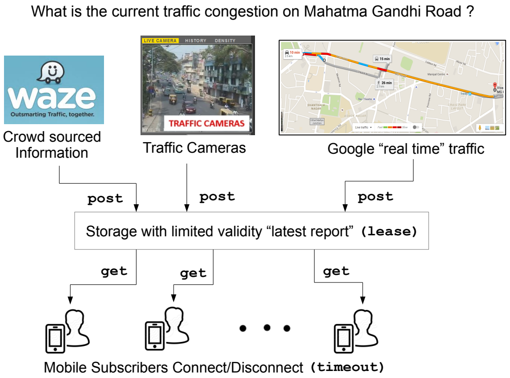
lease (s)	$\delta_{\texttt{get}}$ (s)	**Simulation**	**Measurement**
0	exponential(20)	0.65	0.717
0	exponential(40)	0.35	0.42
10	exponential(20)	0.75	0.778
10	exponential(40)	0.48	0.554
40	exponential(20)	0.93	0.91
40	exponential(40)	0.75	0.81

Success Rate Comparison

- Compare the results of simulated and measured success rates
- Absolute deviation between the two is no more than 10%
- Deviation may be due to network delays, buffering at each entity (poster, getter, intermediate entity)
- Designers can rely on our simulation model for tuning their system parameters

Outline

Conclusions



- `lease`=10 seconds, `timeout`=10 seconds:
  Success Rate 65% and Latency within 8 sec. ($\mathbb{P} = 0.63$)
- `lease`=10 seconds, `timeout`=20 seconds:
  Success Rate 80% and Latency within 4 sec. ($\mathbb{P} = 0.77$)

Conclusions

- Unified timing analysis across heterogeneous middleware paradigms using XSB
- Demonstrated the effect of varying `lease` and `timeout` periods on success rates/latency
- By leveraging the timing analysis, designers can accurately set constraints to ensure high success rates for transactions
- Included stochastic behavior without more sophisticated formal notations (probabilistic timed automata) - future work

Thank you

TATA CONSULTANCY SERVICES

Experience certainty.

informatics mathematics