DATE 16
DESIGN, AUTOMATION & TEST IN EUROPE
14 – 18 March, 2016 · ICC · Dresden · Germany
The European Event for Electronic
System Design & Test

Quantitative Timing Analysis of UML Activity Digrams using Statistical Model Checking

Fan Gu[1], Xinqian Zhang[1], Mingsong Chen[1],
Daniel Grosse[2] and Rolf Drechsler[2]
[1] *Institute of CS & SE, East China Normal University, China*
[2] *Institute of Computer Science, University of Bremen, Germany*

華東師範大學
EAST CHINA NORMAL UNIVERSITY

Universität Bremen

Outline

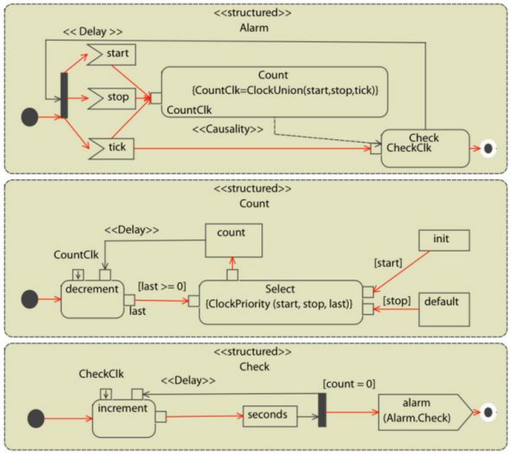
- Introduction
- Preliminary Knowledge
  - Variation-aware Construction of NPTA
  - UPPAAL-SMC Based Evaluation
- Our Quantitative Timing Analysis Approach
  - Extension of UML Activity Diagrams
  - NPTA Model Generation
  - Property Generation & Quantitative Analysis
- Experimental Results
- Conclusion

Outline

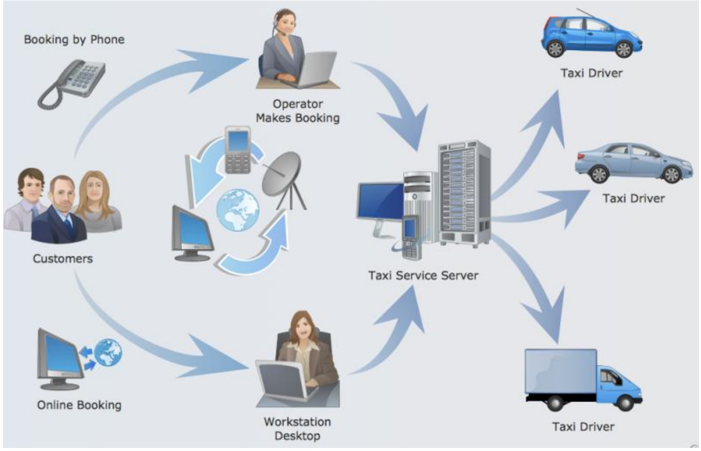
- Introduction
- Preliminary Knowledge
  - Variation-aware Construction of NPTA
  - UPPAAL-SMC Based Evaluation
- Our Quantitative Timing Analysis Approach
  - Extension of UML Activity Diagrams
  - NPTA Model Generation
  - Property Generation & Quantitative Analysis
- Experimental Results
- Conclusion

Modeling with UML Activity Diagrams

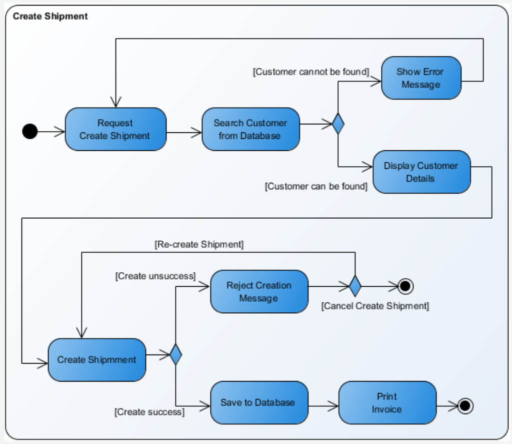
- Based on *Petri-net* semantics, activity diagrams are widely used in modeling concurrent behaviors of system designs.
  - Easier to understand than text
  - Friendly for both HW and SW designers
  - Support complex functional checking and timing verification



Real-Time and Embedded Systems



Service Workflow

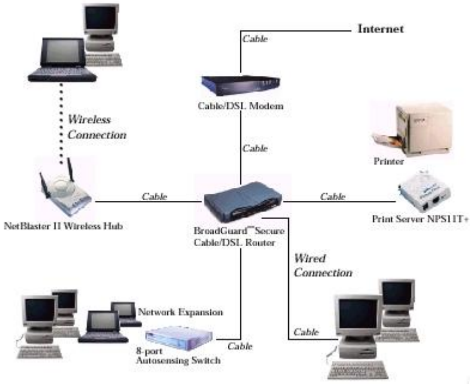


Business rules and operations

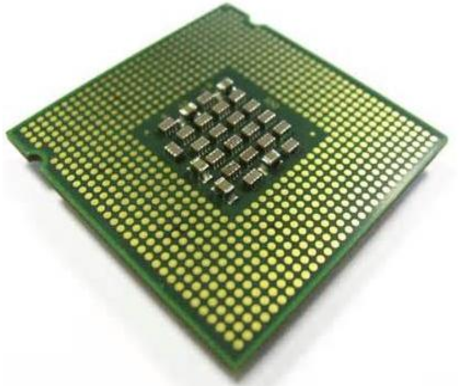
Timing Analysis of Activity Diagrams

- Due to increasing interactions with uncertain environment, the timing of system behaviors becomes hard to be predicted.

Human-in-the-Loop



Network Delay



Device Variations

- Within an uncertain environment, activity diagrams designers would like to ask the question *“What is the probability that a specific scenario can be complete within a time limit?”*.
- Unfortunately, few of existing approaches can model and reason the timing of activity diagram behaviors under variations (e.g., user-input, action execution time).

Limitations & Challenges

- Approach to analyze activity diagrams
  - Model checking based methods
    - Consistency checking ***(Eshuis, TOSEM 2006; Hilken et al., FDL 2014)***
    - Timing verification ***(Li et al., UML 2001; Das et al., ASPEC 2006)***
  - Model-driven testing approaches
    - Gray-box testing ***(Wang et al., APSEC 2004)***
    - Directed testing ***(Chen et al., GLSVLSI 2008; Chen et al., DAES 2010)***

Challenges

i) How to accurately model system behaviors under various kinds of variations?

ii) How to enable quantitative reasoning of critical functional and performance requirements?

modeling (e.g., uniform distribution)

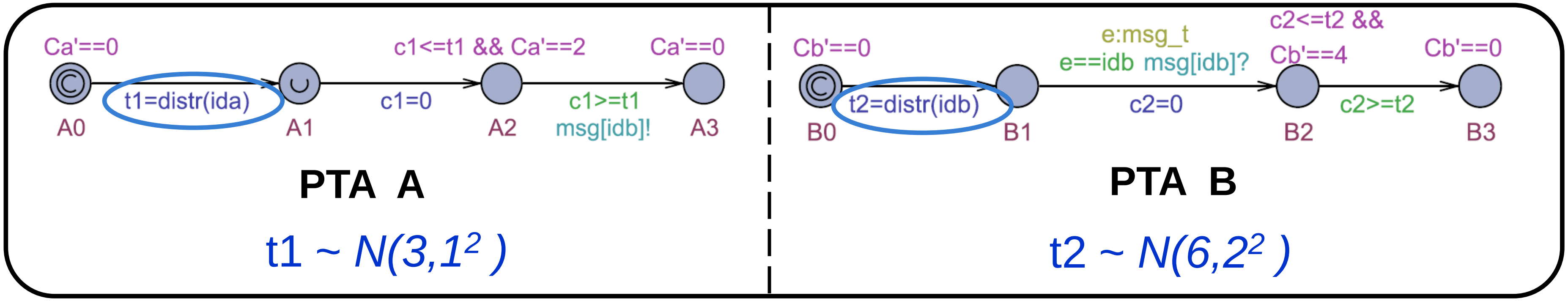
- Lack of automated tools to enable the quantitative reasoning about the performance metrics

Outline

- Introduction
- Preliminary Knowledge
  - Variation-aware Construction of NPTA
  - UPPAAL-SMC Based Evaluation
- Our Quantitative Timing Analysis Approach
  - Extension of UML Activity Diagrams
  - NPTA Model Generation
  - Property Generation & Quantitative Analysis
- Experimental Results
- Conclusion

Variation-Aware Construction of NPTA

- **NPTA - Network of Priced Timed Automata**
- **An NPTA instance, (A | B)**



PTA A

$t1 \sim N(3,1^2)$

PTA B

$t2 \sim N(6,2^2)$

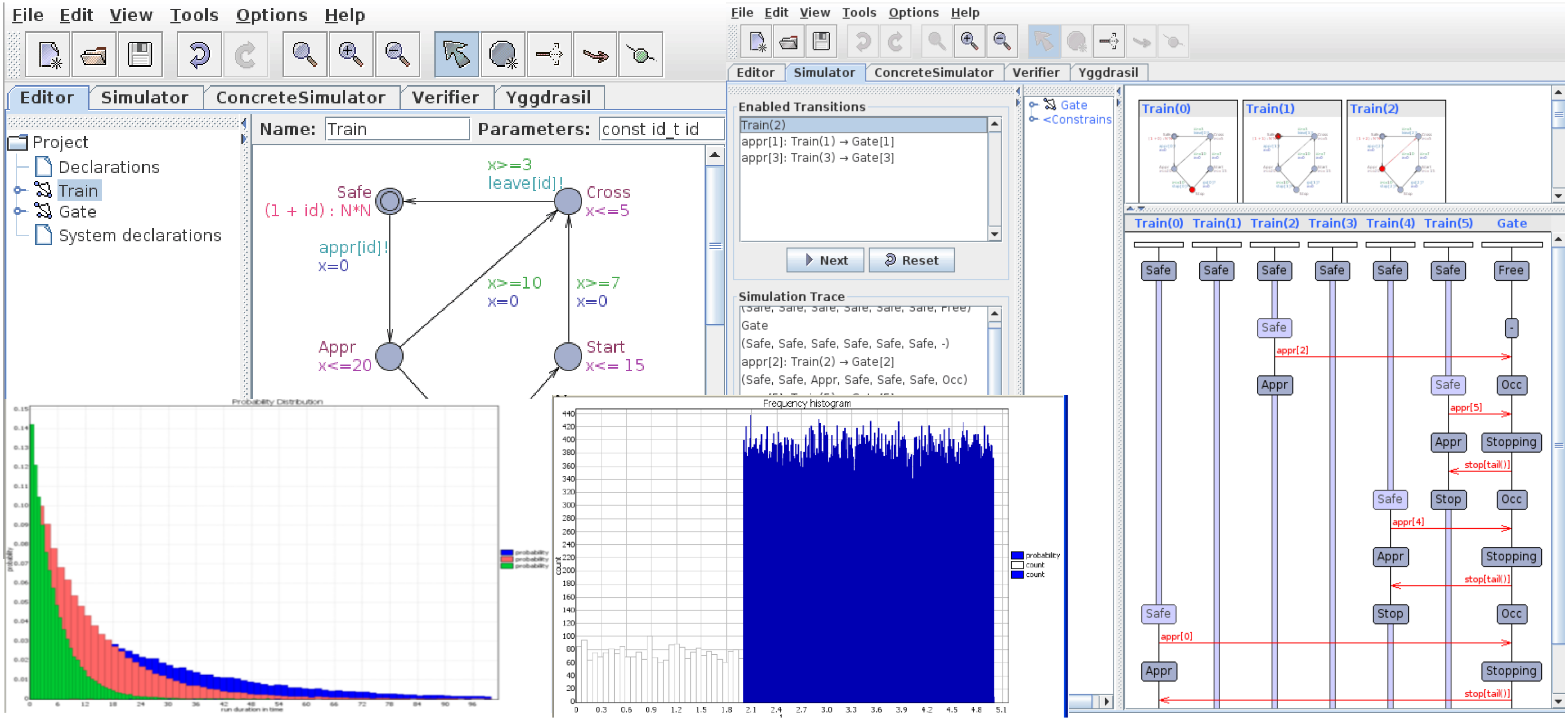
Time of reaching ($A3$, $B3$) ~ $N(9,1^2+2^2)$.

- **A possible behavior of the NPTA (A|B)**

$$((A_0,B_0),[c_1=0,c_2=0,C_a=0,C_b=0]) \xrightarrow{0}$$
$$((A_1,B_1),[c_1=0,c_2=0,C_a=0,C_b=0]) \xrightarrow{0}$$
$$((A_2,B_1),[c_1=0,c_2=0,C_a=0,C_b=0]) \xrightarrow{2.5} \xrightarrow{msg[idb]!}$$
$$((A_3,B_2),[c_1=2.5,c_2=0,C_a=5,C_b=0]) \xrightarrow{5.1}$$
$$((A_3,B_3),[c_1=7.6,c_2=5.1,C_a=5,C_b=20.4]) \xrightarrow{\cdots} \ldots$$

UPPAAL-SMC

- UPPAAL-SMC versus formal model checking
  - Based on simulation, thus requiring far less memory and time
  - Allow high scalable validation approximation
  - Support quantitative performance analysis
- Applications: Real-time systems, Smart building, Biology, …

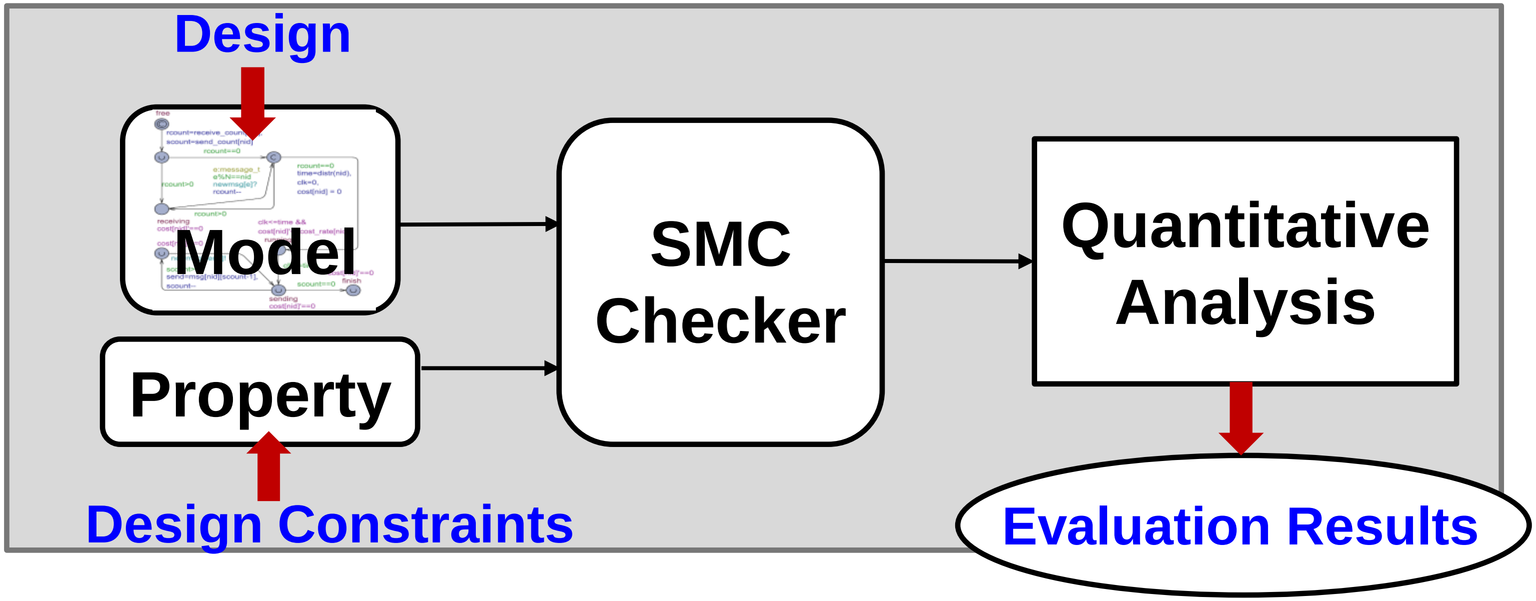


```
Train(2)
appr[1]: Train(1) → Gate[1]
appr[3]: Train(3) → Gate[3]
```

```
(Safe, Safe, Safe, Safe, Safe, Safe, Free)
Gate
(Safe, Safe, Safe, Safe, Safe, Safe, -)
appr[2]: Train(2) → Gate[2]
(Safe, Safe, Appr, Safe, Safe, Safe, Occ)
```

UPPAAL-SMC Based Evaluation

- Our quantitative analysis is based on UPPAAL-SMC, which is effective for checking large stochastic systems



Design

Model

Property

Design Constraints

SMC Checker

Quantitative Analysis

Evaluation Results

- Query formats supported by UPPAAL-SMC
  - Qualitative check: ***Pr [time <= bound] (<> expr) >= p***
  - Quantitative check: ***Pr [time <= bound] (<> expr)***
  - Probability comparison:

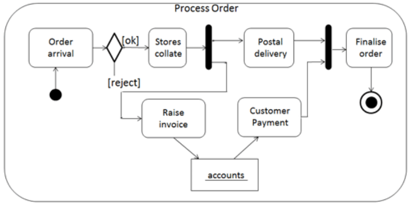
Pr [time1 <= bound1] (<> expr1) >= Pr [time2 <= bound2] (<> expr2)

Outline

- Introduction
- Preliminary Knowledge
  - Variation-aware Construction of NPTA
  - UPPAAL-SMC Based Evaluation
- Our Quantitative Timing Analysis Approach
  - Extension of UML Activity Diagrams
  - NPTA Model Generation
  - Property Generation & Quantitative Analysis
- Experimental Results
- Conclusion

Problem Definition

UML Activity Diagrams



Variation Information

- Network delay
- Execution time variation
- User input variation

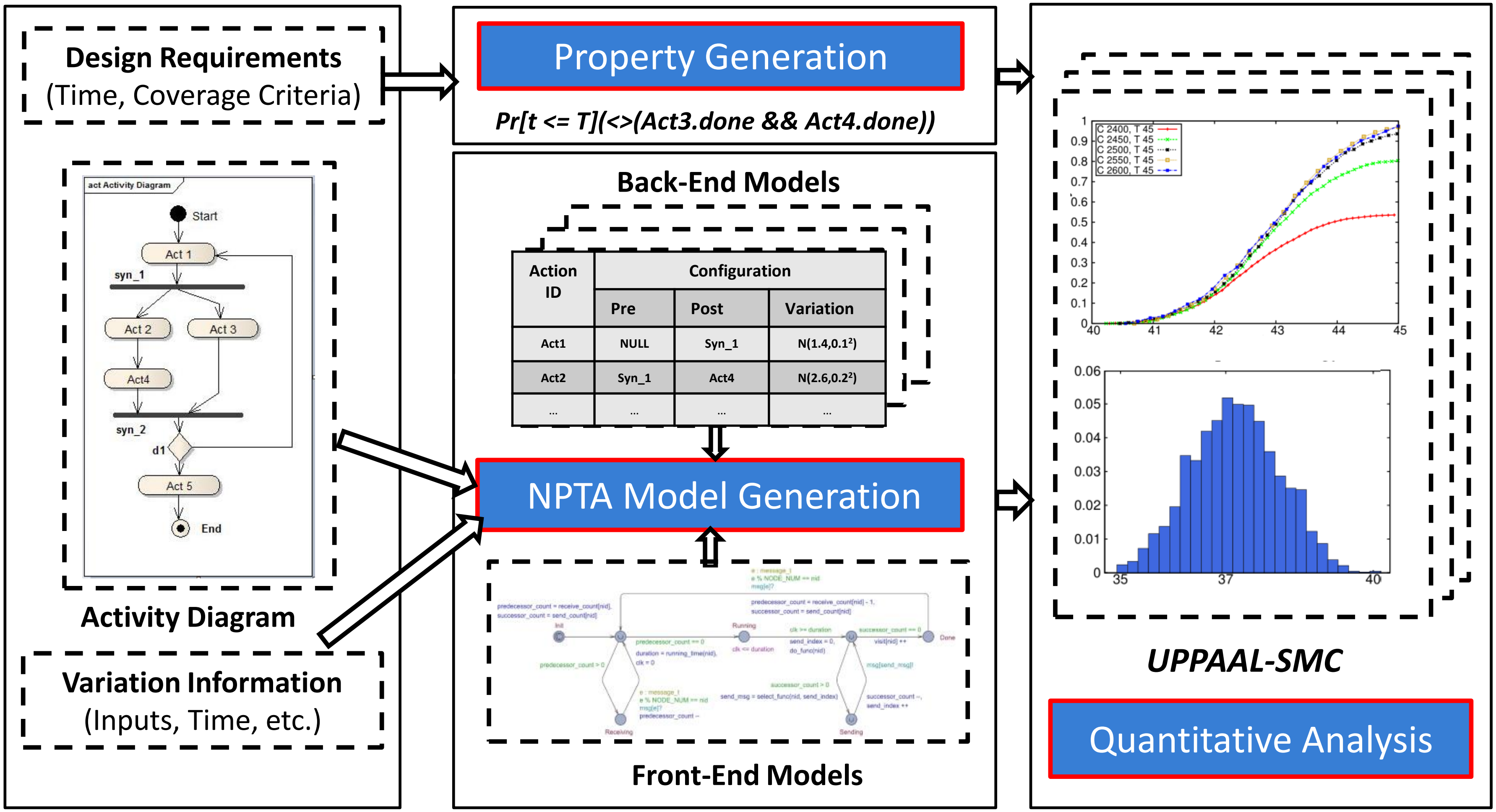
Design Requirements

- Response time
- Functional Scenarios

Quantitative Evaluation Framework

Timing Analysis Results

Our Framework



Design Requirements
(Time, Coverage Criteria)

Activity Diagram

Variation Information
(Inputs, Time, etc.)

Property Generation

Pr[t <= T](<>(Act3.done && Act4.done))

Back-End Models

Action ID	Configuration		
	Pre	Post	Variation
Act1	NULL	Syn_1	$N(1.4,0.1^2)$
Act2	Syn_1	Act4	$N(2.6,0.2^2)$
…	…	…	…

NPTA Model Generation

Front-End Models

UPPAAL-SMC

Quantitative Analysis

- All the three steps are fully automated.

Graph-Based Notations

- Actions (i.e., functional operations) and activities which indicate a collection of correlated actions

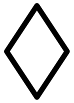
Action

Activity

- Control nodes and flows (indicating the execution order of actions)
  - Control Nodes

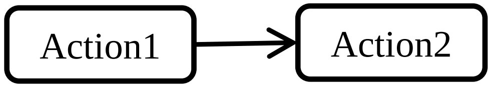
Initial

Final

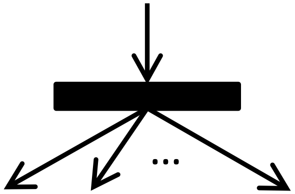


Decision/Merge

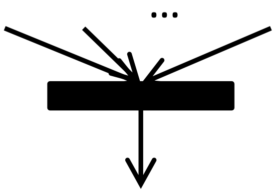
  - Control Flow



Flow edge

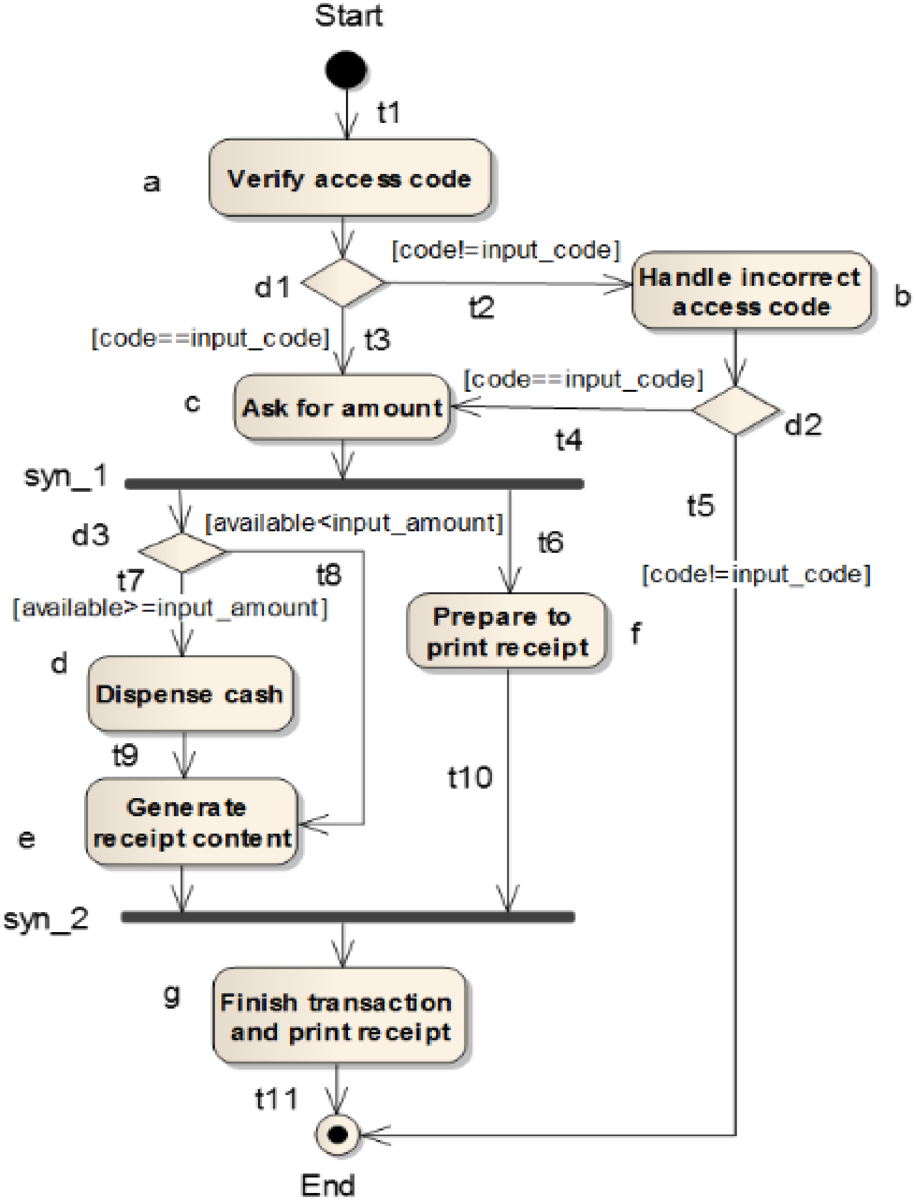


Fork



Join

Activity Diagram Annotation



1. Actions denote operations

e.g., action *d*, i.e., *Dispense_cash*

2: Transitions denote control flows between actions

e.g., Transition *t7* with guard [amount >= available]

3. A run denotes a complete concurrent execution

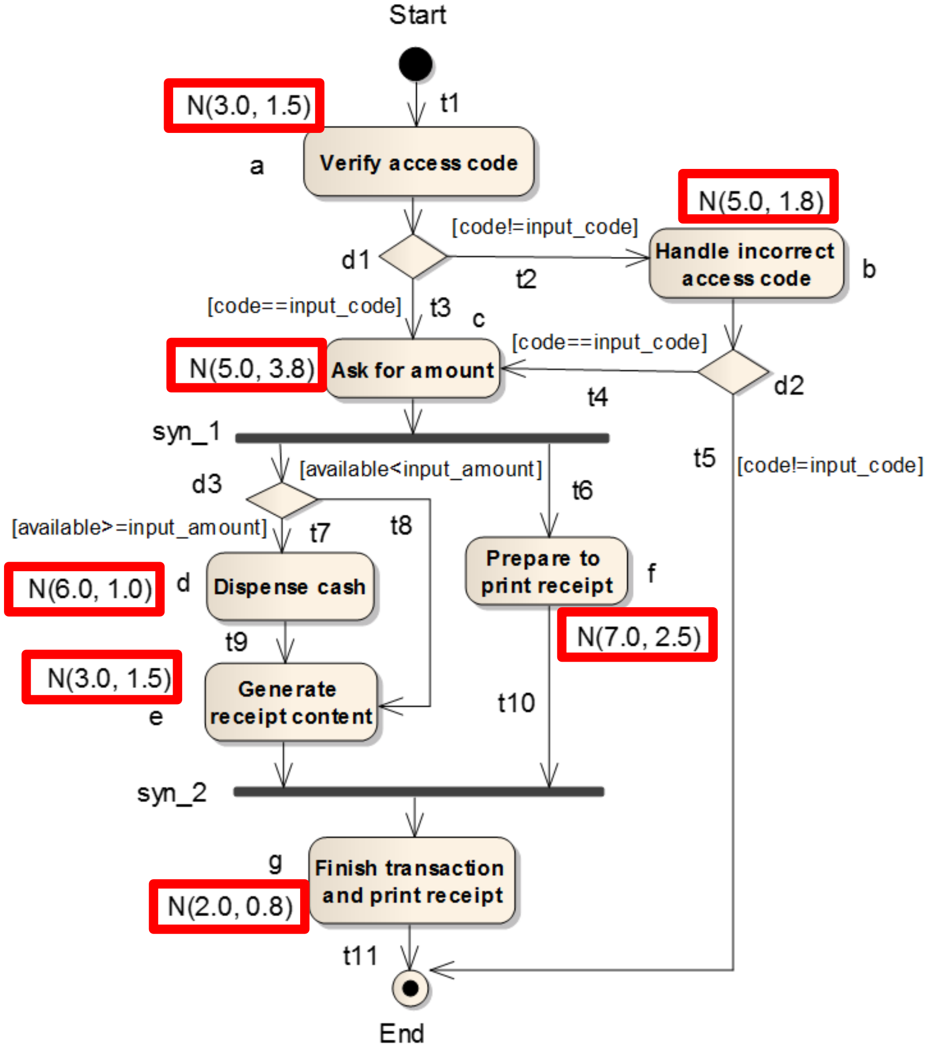
e.g., {Start} -> {a}->{c} -> {d,f}->{e,f} -> {g} -> {End}

Extended Activity Diagrams

User Input:

Input_amount ~ N(500,50)

input_code ~ {“ab”, “abc”, …}



- User inputs are defined following some distributions
- Each operation is assigned with a time distribution, e.g., action d follows normal distribution N(6,1.0)
- Each action corresponds to an operation function
- Distribution information is saved textually as UML notes

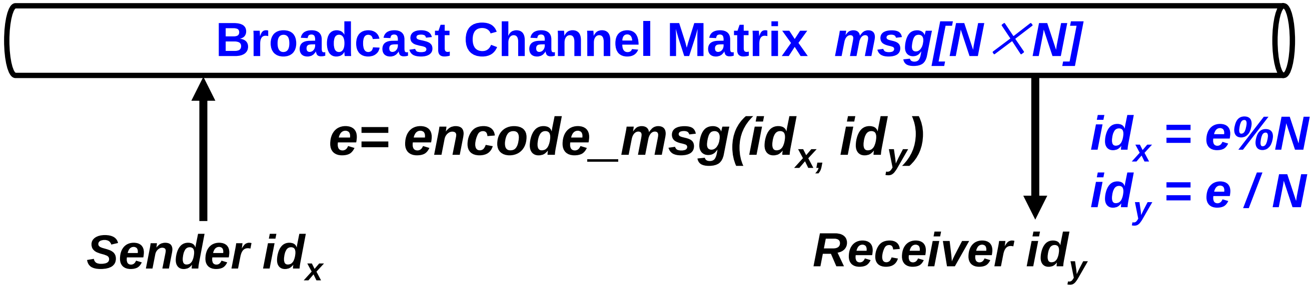
NPTA Model Generation

- A back-end configuration contains all the information of variations, synchronization and node operations for an activity diagram (with *N* nodes).
  - Activity diagrams are abstracted to DAGs with nodes (action nodes and control nodes) and edges (control flows).
  - Synchronization bars are not modeled explicitly. We assume that a node can be executed only when all its precedent nodes are complete.
- Back-end configuration of variation information
  - For input variables, the configuration defines their value distributions, and their random values are generated in the initial action
  - Action time distributions are save in *distribution[N][m].*
    - E.g., if *action i* follows normal distribution of, *distribution[i][0]* indicates its expected execution time, and *distribution[i][1]* stores the standard deviation.

NPTA Model Generation

- **Action synchronization via channel communication**
  - UPPAAL-SMC communicates via broadcasting
  - Point-to-Point communication encoding using the formula

$$encode\_msg(id_x, id_y) = id_x \times N + id_y$$



Broadcast Channel Matrix ***msg[N×N]***

$e = encode\_msg(id_x, id_y)$

$id_x = e\%N$

$id_y = e / N$

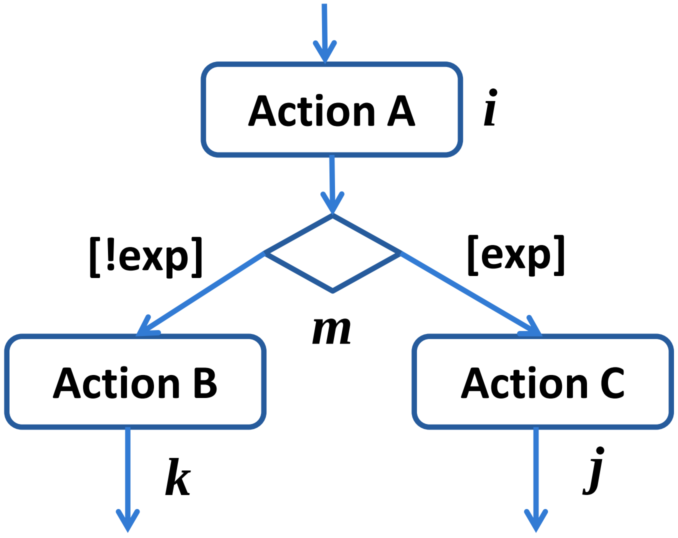
Sender id_x

Receiver id_y

- **Back-end configuration of synchronization**
  - Flow edges indicate the unidirectional communication
  - Instead of creating an urgent channel array *msg_graph[N][N]*, we use a two-dimensional array *msg_graph[N][Max_Out]*, where *msg_graph[i][j]* indicates the j_{th} channel from node *i*.

NPTA Model Generation

- **Back-end configuration of node operation**
  - Action node function: There is an action function for action with ID *nid* named *act_func_nid()*, which will be called by a uniform function *do_func(nid)*.
  - Branch node function: For each control node (i.e., decision or merge), we create a branch function *br_func_nid()*, which will be called by a uniform function *select_func(nid)*.

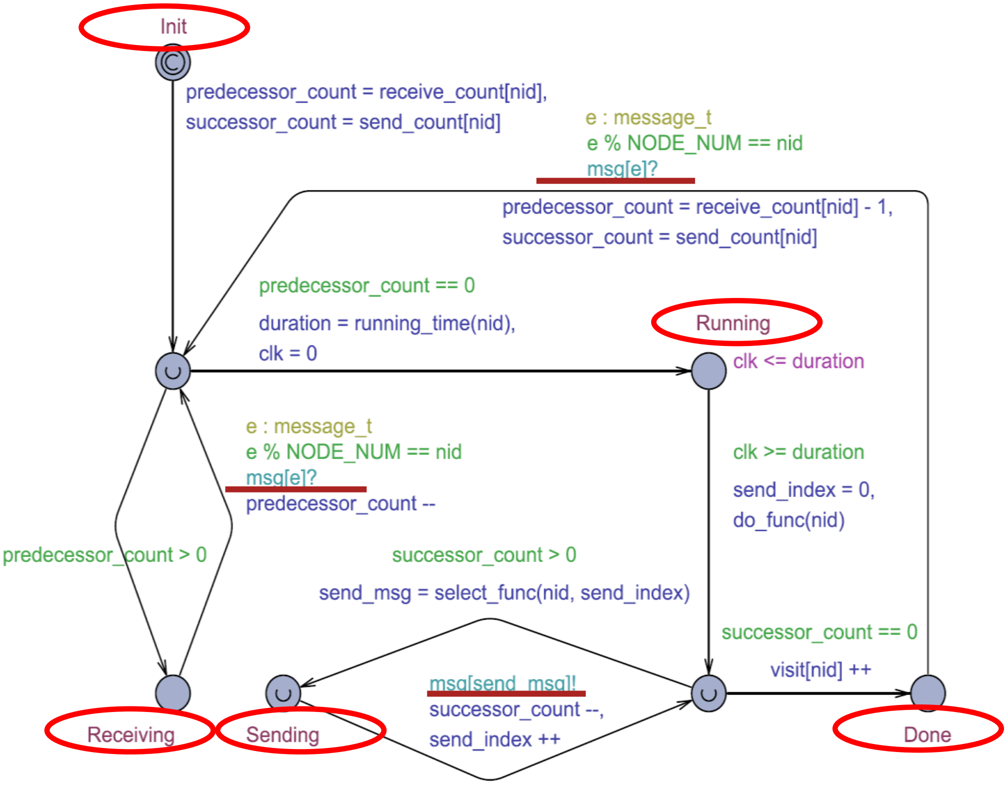


```
message_t  br_func_m(id_t nid){
      if (exp) return msg_graph[nid][0];  // channel to action c
      if (!exp) return msg_graph[nid][1]; // channel to action b
      else return -1;
 }
message_t  br_func_n(id_t nid);
……
message_t  select_func(id_t nid){
   if (nid==m)  return br_func_m(nid);
   if  (nid==n)  return br_func_n(nid);
   ……
   return -1;
 }
```

NPTA Model Generation

Front-end Model for Node

(action node & control node)



- **Initial state**
  - ❑ The beginning of a task
- **Receiving state**
  - ❑ Tries to get notification messages from all the predecessors
- **Running state**
  - ❑ All predecessors finished
  - ❑ Current task is executing
- **Sending state**
  - ❑ Notify all successive tasks about its completion
- **Done state**
  - ❑ The completion of a task

Property Generation & Evaluation

"What is the probability that a functional scenario S can happen or complete within a time limit T?"

Pr [<= T] (<> S.done)

- ***[<= T]*** indicates the time limit is T
- **<>*S*** *checks* whether scenario S can be fulfilled eventually.
- ***S.done*** indicates the completion of scenario ***S***
- Based on parameters ε (probability uncertainty) and α (probability of false negatives) , stochastic runs are generated to obtain an approximate interval [p- ε,p+ ε] with a confidence 1- α

Coverage-Oriented Property Generation

Supports three kinds of performance queries obtained from the structural information of activity diagrams.

- Action queries
  - ☐ act_i can be visited at least k times and the last state is *sta*

$$Pr\ [<= T]\ (<>\ act_i.sta\ \&\&\ visit[i]>=k)$$

- Interaction queries
  - ☐ The actions with specified states can happen simultaneously

$$Pr\ [<= T]\ (<>\ act_i.sta_1\ \&\&\ act_j.sta_2)$$

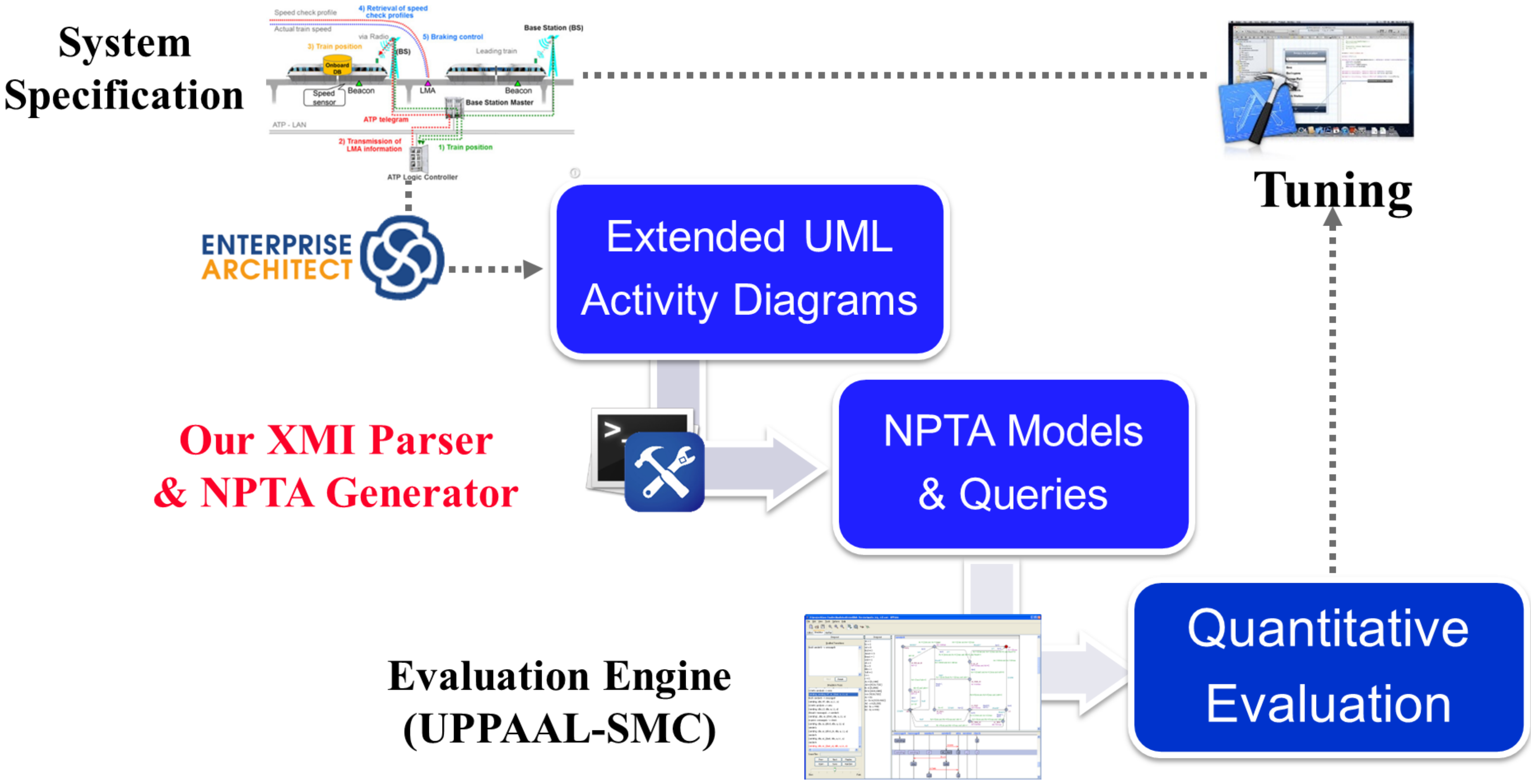
- Run Queries
  - ☐ The run can complete within a time limit T

$$Pr\ [<= T]\ (<>\ act_{i1}.done\ \&\&\ act_{i2}.done\ \&\&\ldots\&\&\ act_{in}.done\ \&\&\ visit[1]>=k_1\ \&\&\ \ldots\ \&\&\ visit[n]>=k_n)$$

Outline

- Introduction
- Preliminary Knowledge
  - Variation-aware Construction of NPTA
  - UPPAAL-SMC Based Evaluation
- Our Quantitative Timing Analysis Approach
  - Extension of UML Activity Diagrams
  - NPTA Model Generation
  - Property Generation & Quantitative Analysis
- Experimental Results
- Conclusion

Tools Chain for Experiment



System Specification

Extended UML Activity Diagrams

Our XMI Parser & NPTA Generator

NPTA Models & Queries

Evaluation Engine (UPPAAL-SMC)

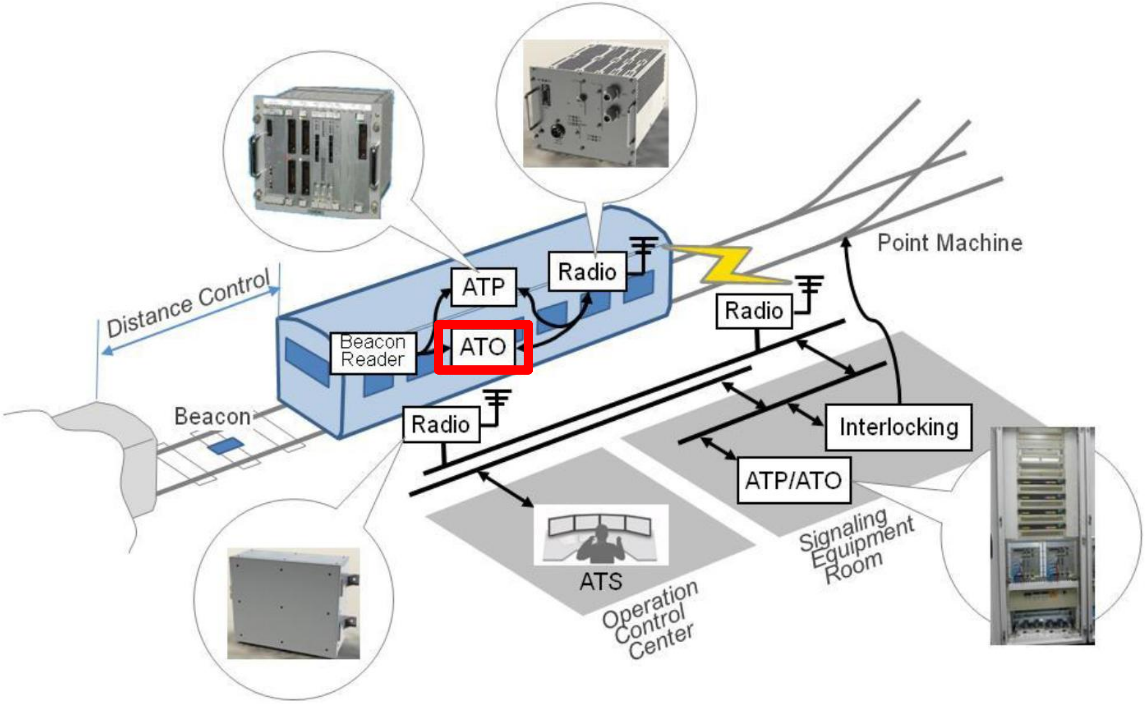
Quantitative Evaluation

Tuning

- ❑ All the experimental results were obtained on a desktop with 3.30GHz AMD CPU and 4GB RAM

Exp. 1 – CBTC ATO Subsystem

- CBTC deals with telecommunications between trains and track equipment. Its subsystem ATO automates operations of trains
- ATO suffers from the delay of communication and the execution time variations of software and hardware components.

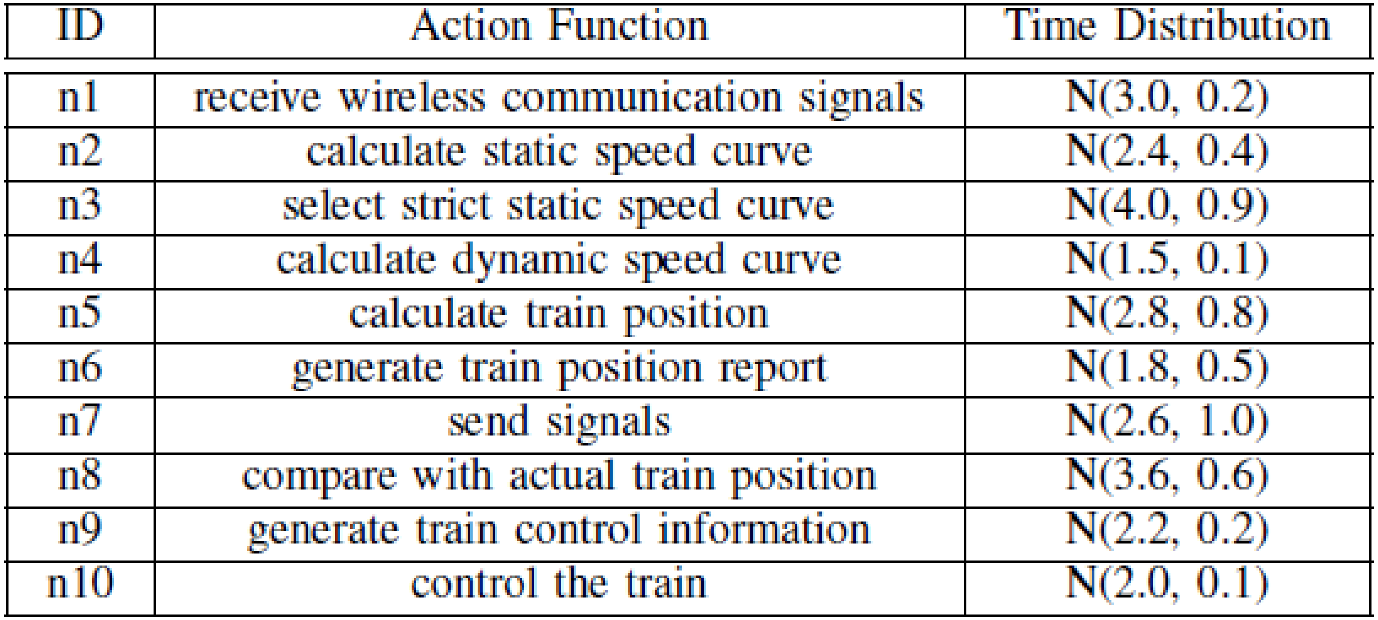


Outline of CBTC system configuration

Source：Hitachi CBTC SIL4 news release

Exp. 1 – CBTC ATO Subsystem

- We focus on analysis of communication delay and execution time variation for ATO (with ε=0.02, α=0.02)
- The activity diagram has 10 action nodes, 2 fork bars and 2 join bars. The functional description and variation information of actions are as follows.

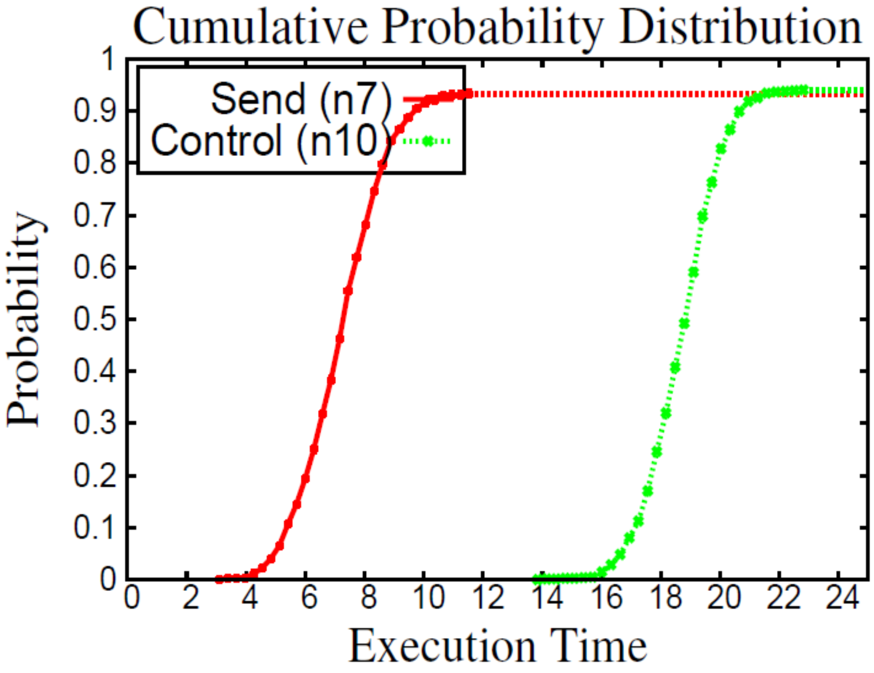


ID	Action Function	Time Distribution
n1	receive wireless communication signals	N(3.0, 0.2)
n2	calculate static speed curve	N(2.4, 0.4)
n3	select strict static speed curve	N(4.0, 0.9)
n4	calculate dynamic speed curve	N(1.5, 0.1)
n5	calculate train position	N(2.8, 0.8)
n6	generate train position report	N(1.8, 0.5)
n7	send signals	N(2.6, 1.0)
n8	compare with actual train position	N(3.6, 0.6)
n9	generate train control information	N(2.2, 0.2)
n10	control the train	N(2.0, 0.1)

Table 1: Execution Time Distributions of ATO Actions

Exp. 1 – CBTC ATO Subsystem

- We use action query to check the probability of an action completion within a time limit
- The evaluation costs around 5 minutes
- We can observe that, after a threshold, the change of the completion probability is quite small!



Query 1: Pr [<= 25] (<> n7.done)

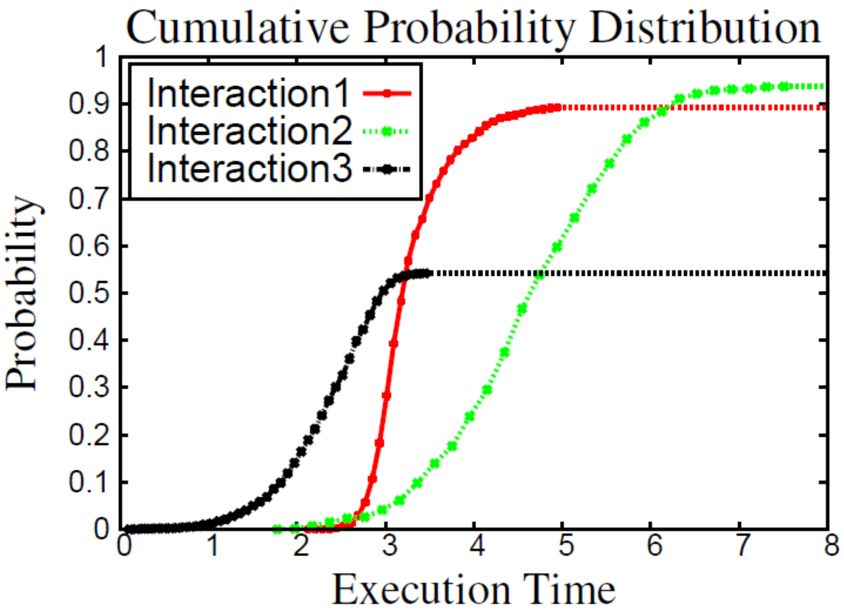
With 890 runs, obtain a probability interval [0.91,0.95] with a confidence 98%

Query 2: Pr [<= 25] (<> n10.done)

With 808 runs, obtain a probability interval [0.92,0.96] with a confidence 98%

Exp. 1 – CBTC ATO Subsystem

- We adopt interaction queries to check correlation between concurrent execution components. Each evaluation costs less than 5 minutes



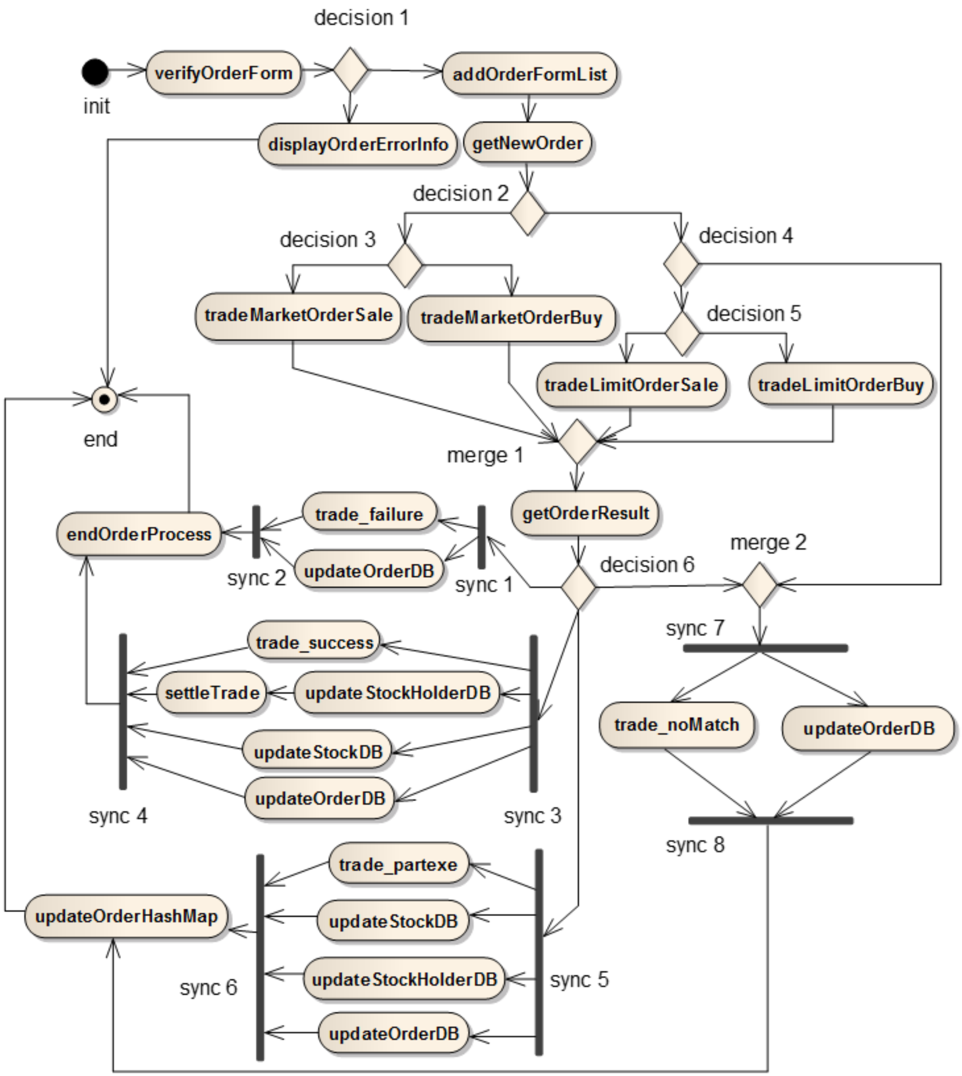
***Scenario 1:* Pr [<= 5] (<> n2.running && n6.running) checks the *overlapped execution* between actions n2 and n6 within 5ms.**

***Scenario 2:* Pr [<= 8] (<> n7.running && n4.receiving) checks the probability that *n7 happens before n4* within 8ms.**

***Scenario 3:* Pr [<= 5] (<> n5.done && n1.running) checks the probability that *n5 completes before the completion of n1* within 5ms**

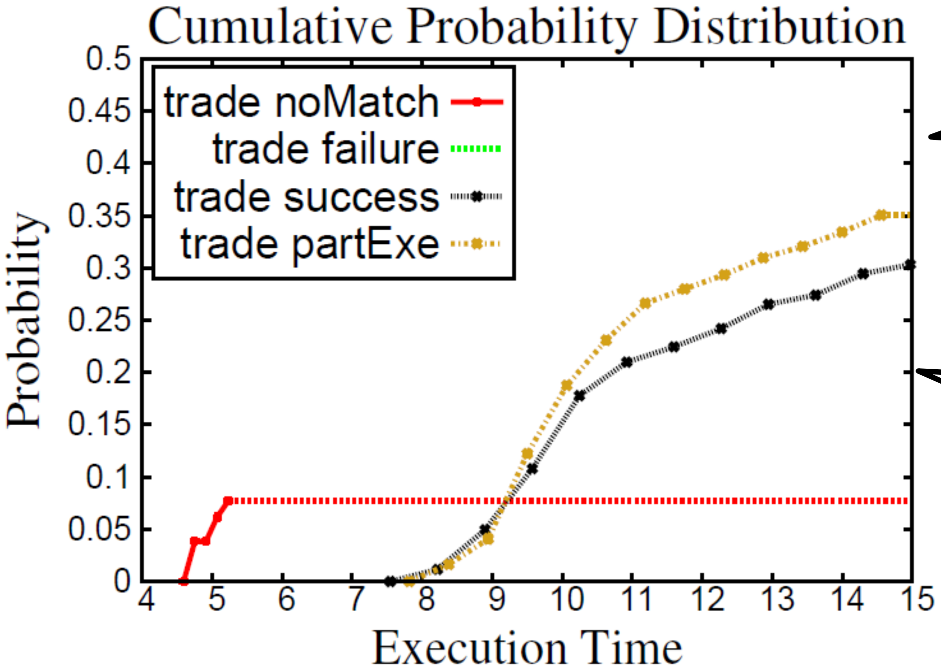
Exp. 2 – OSES Design

- OSES models stock transaction scenarios
- OSES consists of 27 activities, 29 transitions and 8 fork/join bars
- Half orders are buy orders and half orders are sale orders.
- 20% of orders employ market price and 80% orders use limit price.
- We set ε=0.05 and α=0.05



Exp. 2 – OSES Design

- Timing analysis of action completion is important for OSES
  - ❑ Guarantee the proper user experience
  - ❑ Detect performance bottleneck of the system
- We use the action query template *Pr[<=15] (<>act.done)* to check whether *act* can complete within 15 time units. Each query costs around 2-hour SMC simulation time.

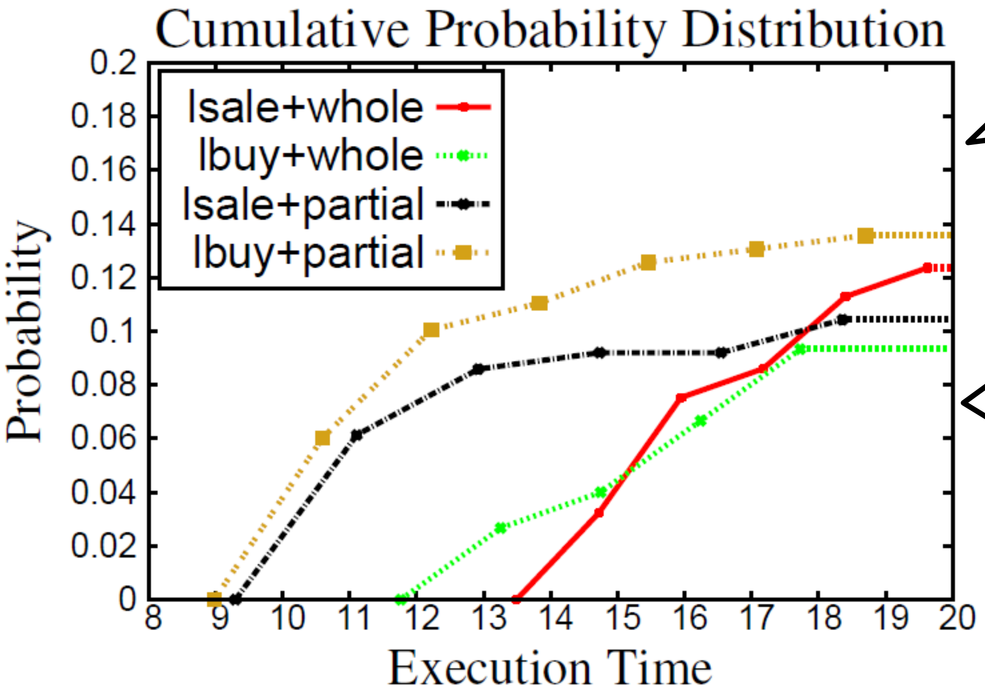


The probability of *noMatch* events is lower than 10%. And *noMatch* can abort the transaction much easier.

The chance of partial execution is a little bit higher than the successful full execution (35% versus 30%).

Exp. 2 – OSES Design

- Since 80% orders are limit orders, our experiment focuses on the quantitative analysis of limit trades.
- We use run queries to check limit sale/buy orders which are categorized as fully traded and partially traded



lbuy+partial **orders achieves the highest probability to complete transactions.**

At time 20, *lsale+whole* has a higher chance to be complete earlier than *lsale+partial*. However, if we set the time limit to be smaller than 18, we will obtain an opposite answer.

Conclusion

- Increasing interactions between systems and surrounded uncertain environment
  - ☐ System behaviors become more stochastic and complex
  - ☐ Correctness and performance cannot be guaranteed
- Proposed an UPPAAL-SMC based quantitative timing analysis framework for activity diagrams
  - ☐ Extend activity diagrams for stochastic behavior modeling
  - ☐ Support complex functional checking and performance queries under variations (e.g., user-input, execution time)
- Comprehensive experimental results demonstrate the efficacy of our approach

Thank you !