Imagining Tests as Interactions

Sagar Sen
Software Engineering Dept., Simula Research Laboratory

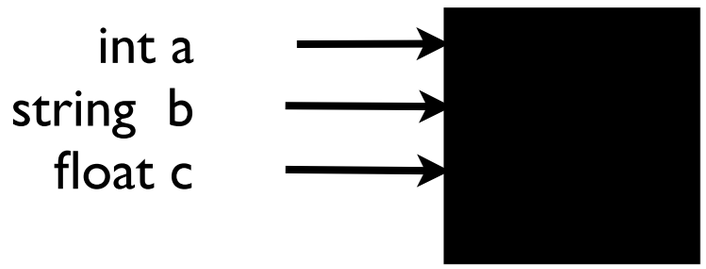
Outline

- **Introduction**
- Case 1: Tests that Cover All T-wise Interactions
- Case 2: Finding Service Level Agreements
- Case 3: Interaction Coverage in Data-intensive Systems
- Case 4: Dynamically Adaptive Vision Systems
- Case 5: Model Checking for Test Generation
- Conclusion

Introduction

Software seen as a black-box

Introduction



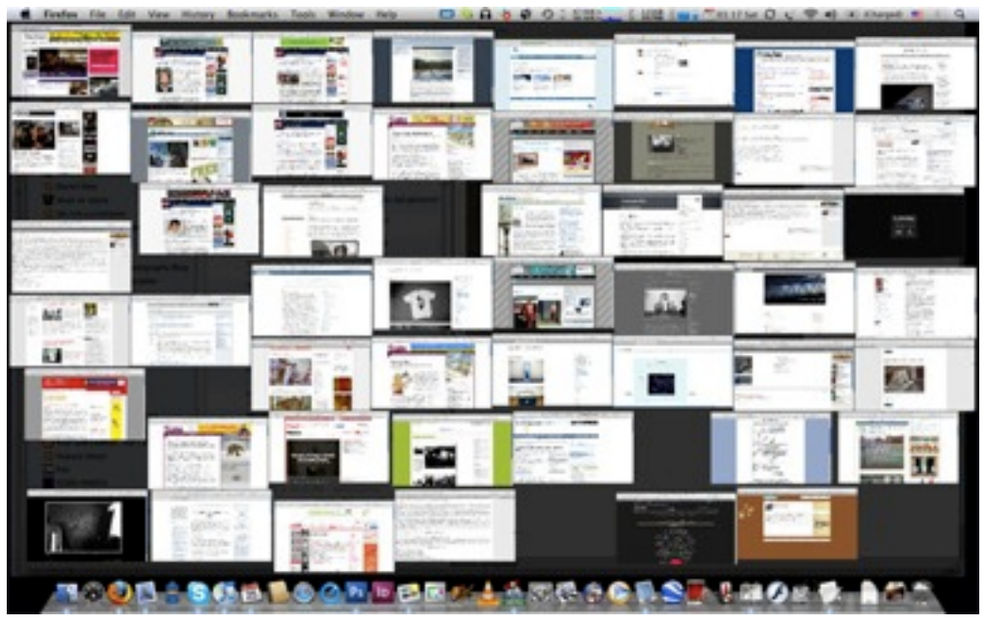
Variability in inputs

Introduction

Variability in ways to configure them

Introduction

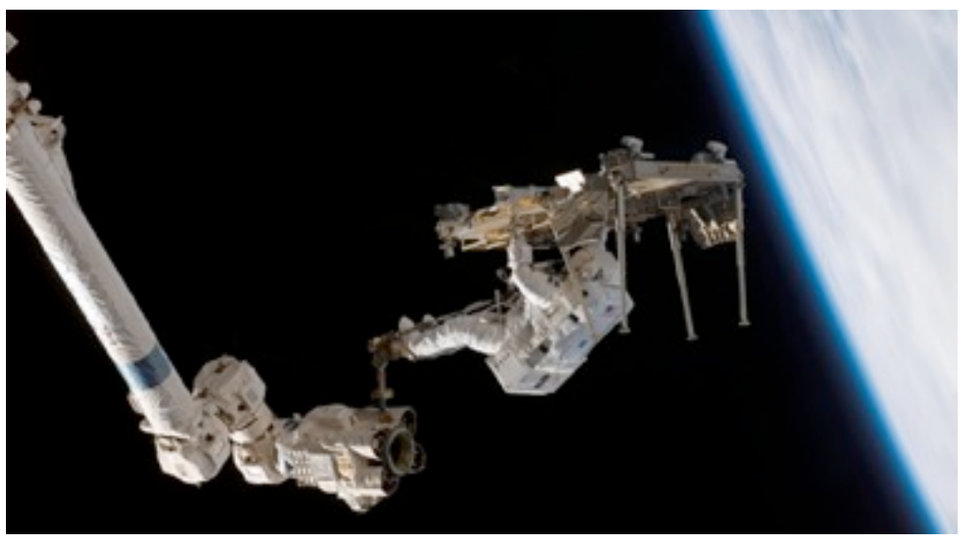
Variability in usage

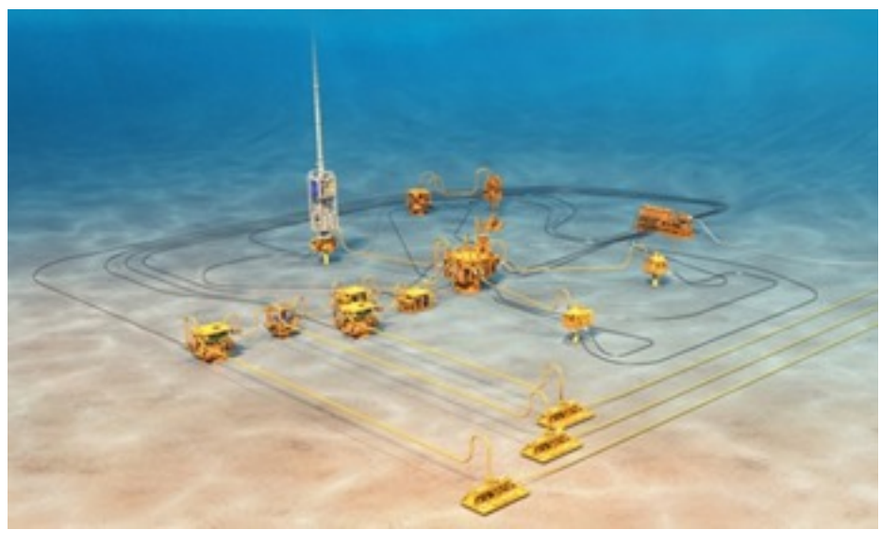


Often linked to the problem of software aging

Introduction

Variability in environment





Space or Subsea

Introduction

How to test software with so much influence of variability?

What does exhaustive testing entail?

Can we reduce the number of tests?

Introduction

What does exhaustive testing entail?

Factor	Operating System	Browser	User's Resolution
1st Level	Win 98	Netscape Navigator	640 x 480
2nd Level	Win ME	Internet Explorer	800 x 600
3rd Level	Win XP		1024 x 768

Introduction

18 Exhaustive Test Cases

Test Case	OS	Browser	Resolution
1	Win 98	Netscape	640 x 480
2	Win 98	Netscape	800 x 600
3	Win 98	Netscape	1024 x 768
4	Win 98	IE	640 x 480
5	Win 98	IE	800 x 600
6	Win 98	IE	1024 x 768
7	Win ME	Netscape	640 x 480
8	Win ME	Netscape	800 x 600
9	Win ME	Netscape	1024 x 768
10	Win ME	IE	640 x 480
11	Win ME	IE	800 x 600
12	Win ME	IE	1024 x 768
13	Win XP	Netscape	640 x 480
14	Win XP	Netscape	800 x 600
15	Win XP	Netscape	1024 x 768
16	Win XP	IE	640 x 480
17	Win XP	IE	800 x 600
18	Win XP	IE	1024 x 768

Introduction

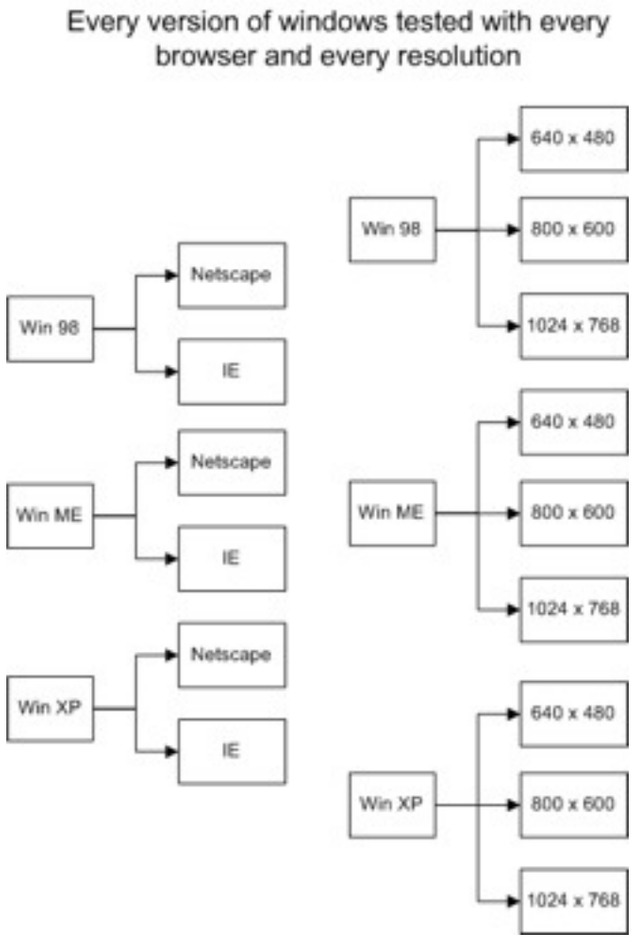
What if the number of factors and their levels increase?

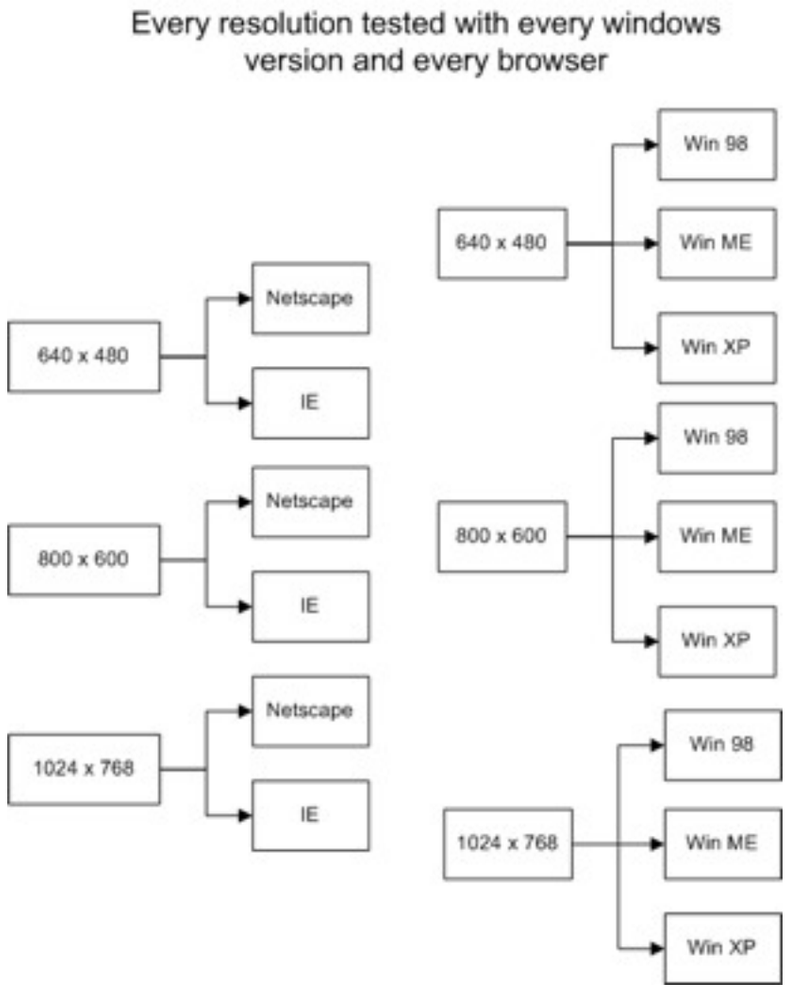
Factors and Levels	All Possible Combinations
5 Factors at 3 levels each	243 Test Cases
6 Factors at 4 levels each	4096 Test Cases
7 Factors at 6 levels each	279,936 Test Cases
10 Factors at 7 levels each	282,475,249 Test Cases

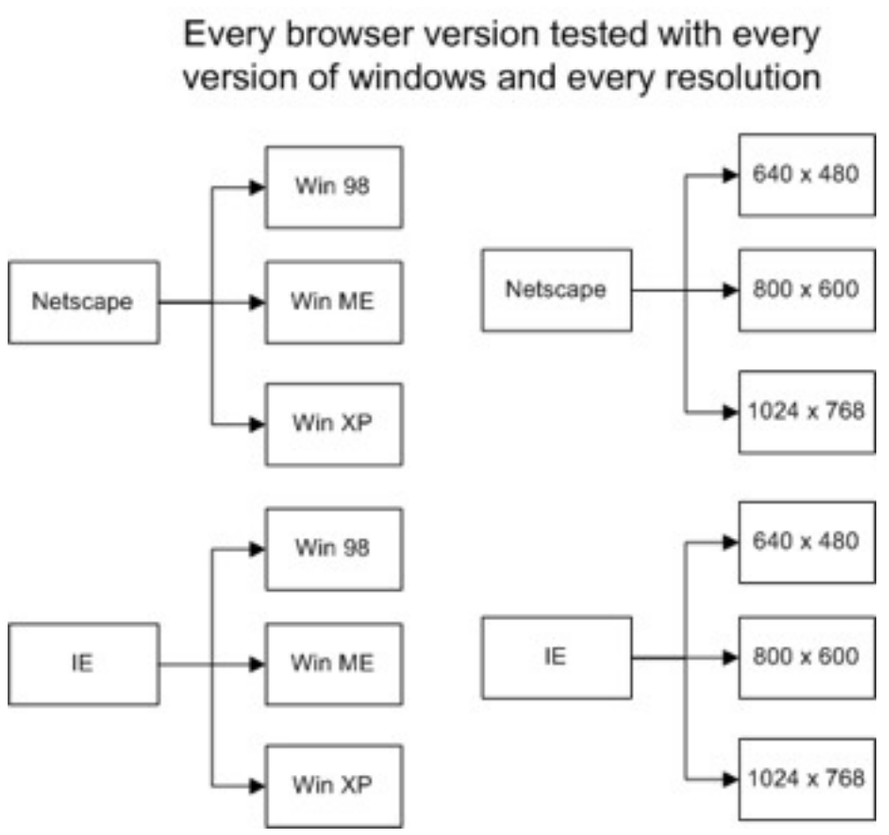
Combinatorial Explosion!

Introduction

What if we cover **all pairs of interactions** between features?







Introduction

9 Pairwise Test Cases

Test Case	OS	Browser	Resolution
1	Win XP	Netscape	1024 x 768
2	Win 98	IE	640 x 480
3	Win XP	IE	800 x 600
4	Win ME	Netscape	640 x 480
5	Win 98	Netscape	800 x 600
6	Win ME	IE	1024 x 768
7	Win XP	IE	640 x 480
8	Win ME	IE	800 x 600
9	Win 98	Netscape	1024 x 768

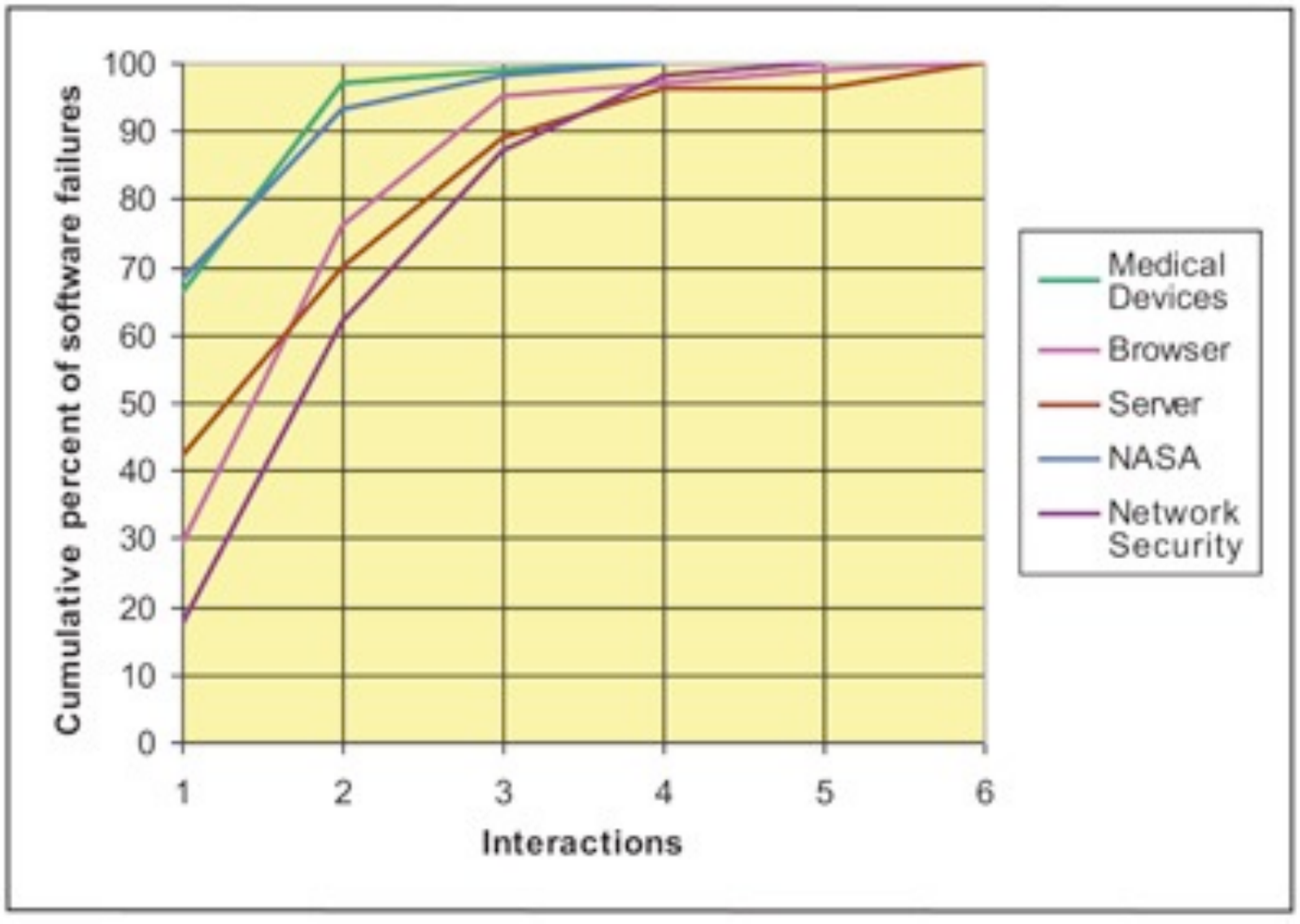
Introduction

Pairwise interaction criteria => Great reduction in test cases

Factors and Levels	All Possible Combinations	Pair wise Combinations
5 Factors at 3 levels each	243 Test Cases	11 Test Cases
6 Factors at 4 levels each	4096 Test Cases	23 Test Cases
7 Factors at 6 levels each	279,936 Test Cases	56 Test Cases
10 Factors at 7 levels each	282,475,249 Test Cases	89 Test Cases

Introduction

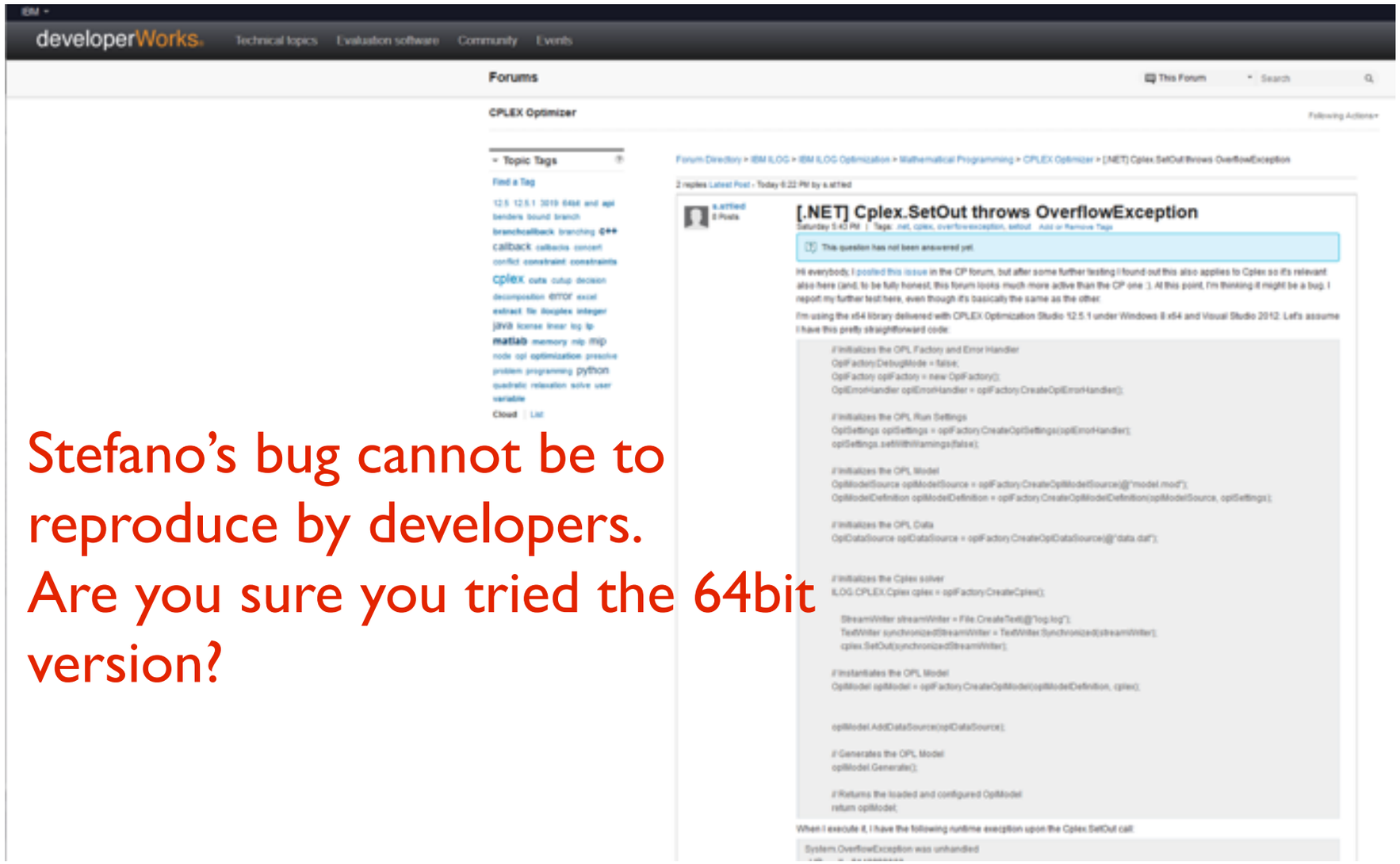
Is pairwise and in general t-wise testing good enough?



"'Combinatorial' Approach Squashes Software Bugs Faster, Cheaper" in *NIST Tech Beat*, Dec. 12, 2007

Introduction

Variability is your enemy but needs embracing!

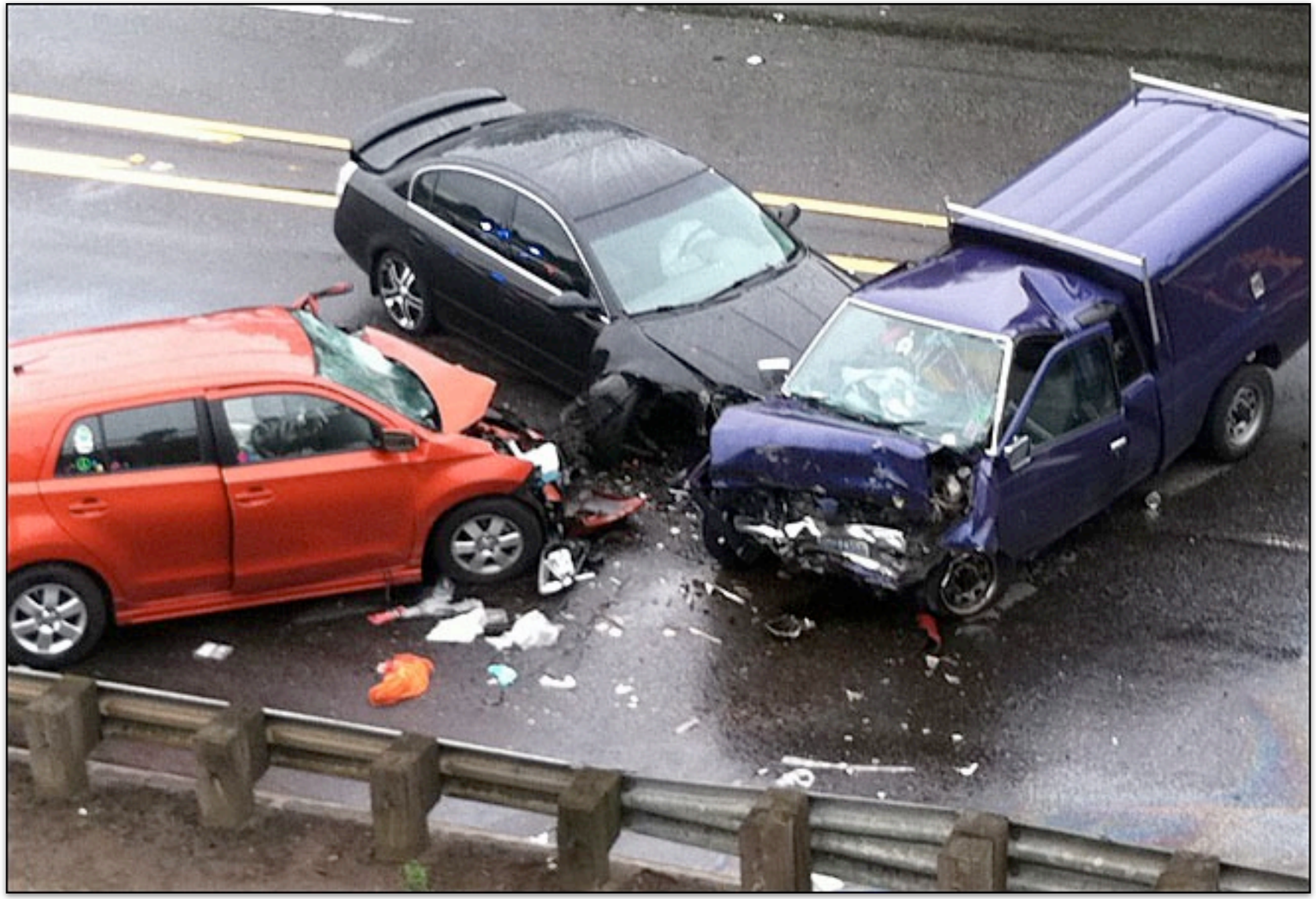


Stefano's bug cannot be to reproduce by developers. Are you sure you tried the 64bit version?

Outline

- Introduction
- **Case 1: Variability in Software and T-wise Tests**
- Case 2: Finding Service Level Agreements
- Case 3: Interaction Coverage in Data-intensive Systems
- Case 4: Dynamically Adaptive Vision Systems
- Case 5: Model Checking for Test Generation
- Conclusion

A Car Crash



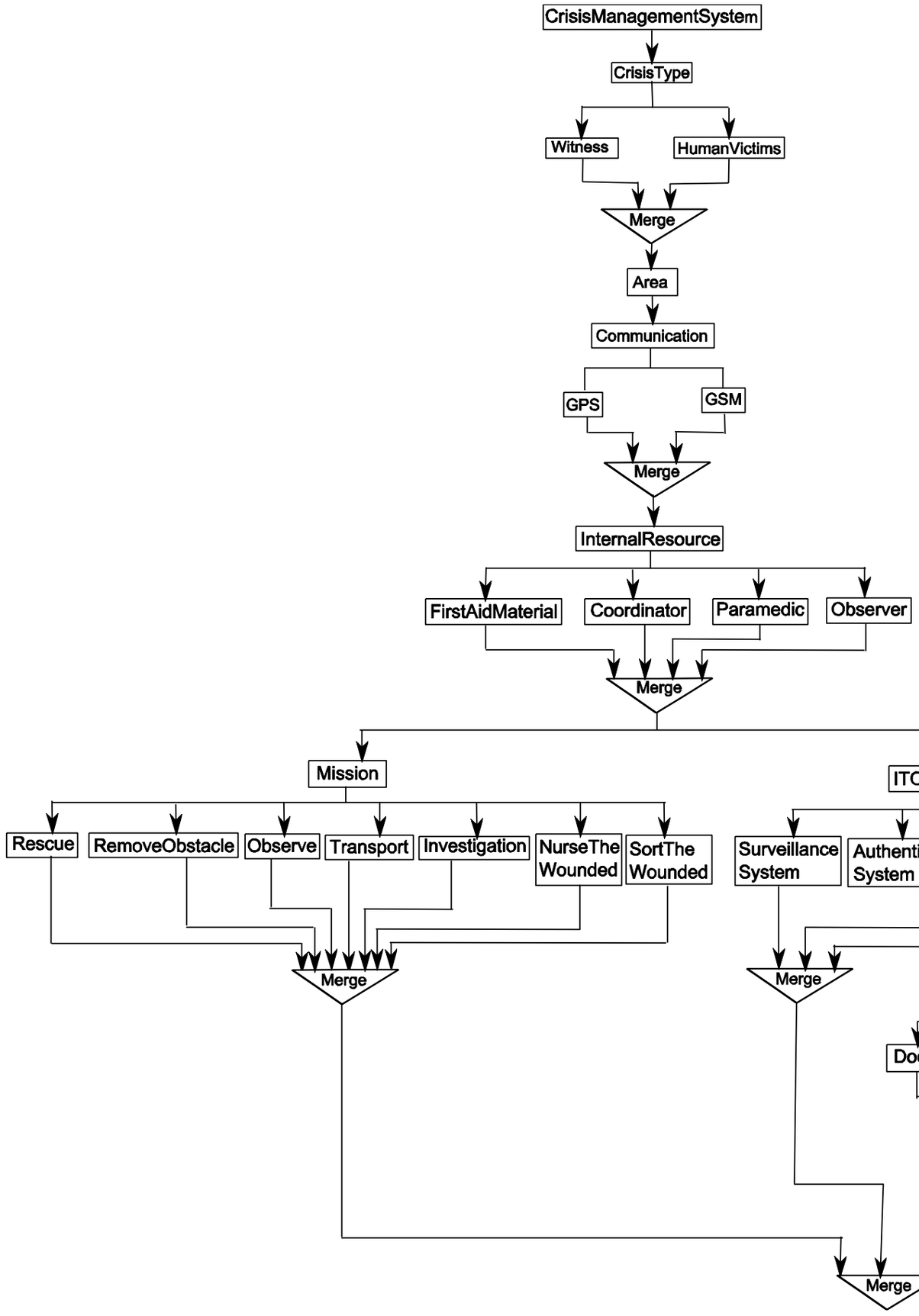
Car Crash Crisis Management Service

Co-ordinate calls to atomic services

1. Ambulance
2. GSM
3. GPS
4. Nursing
5. Public Hospital
6. Doctor
7. Garage Tow Truck
8. Fire
9. Police
10. Authentication system
11. Paramedic

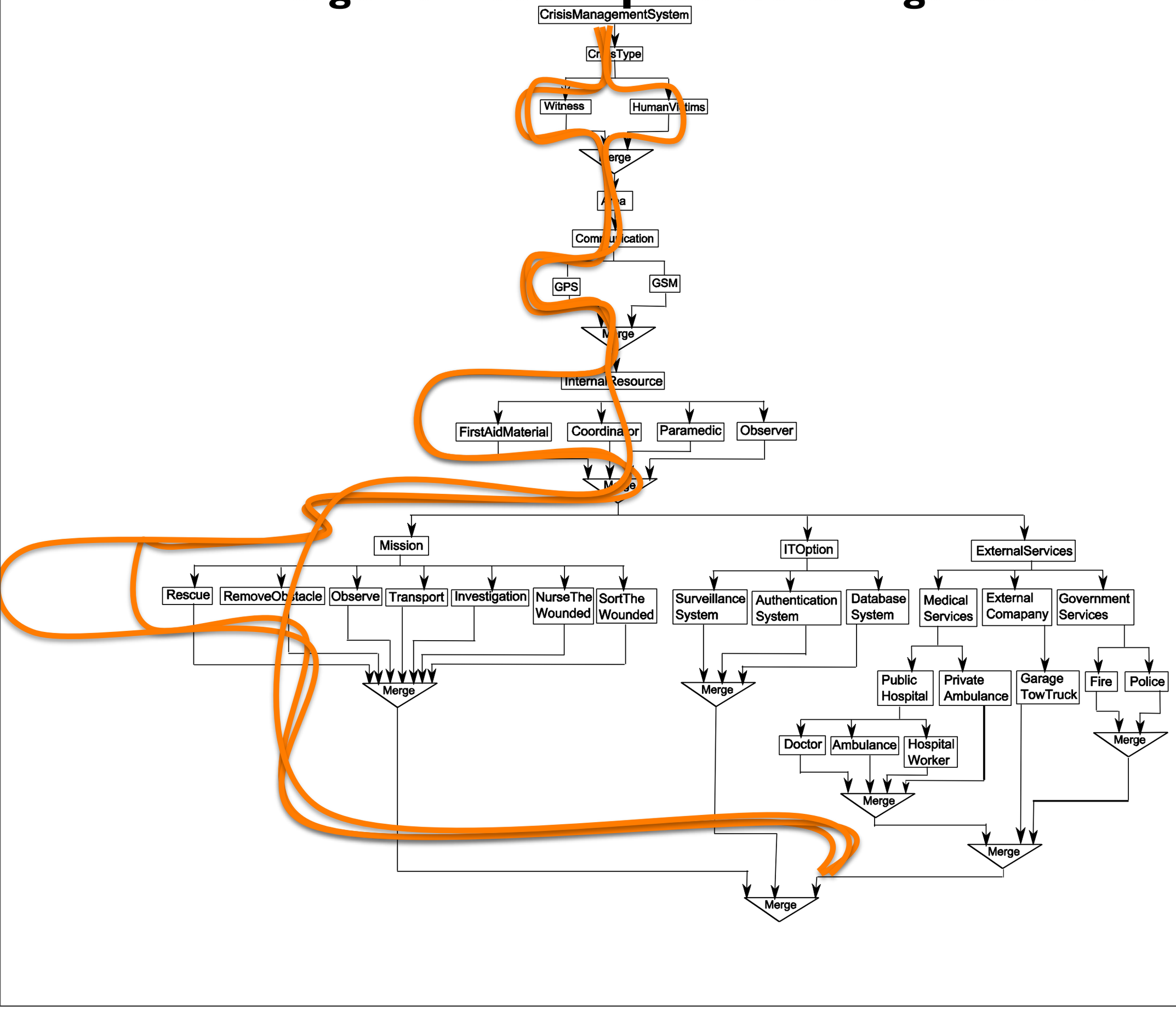
...

Service Level Agreement of Car Crash Crisis Management

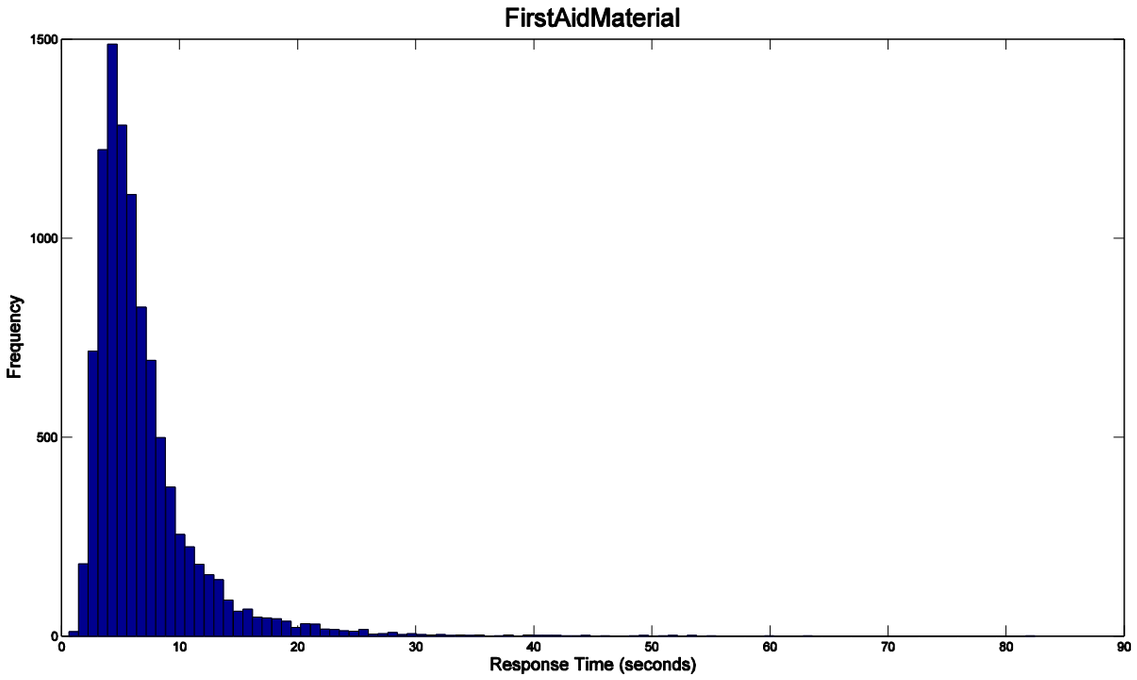


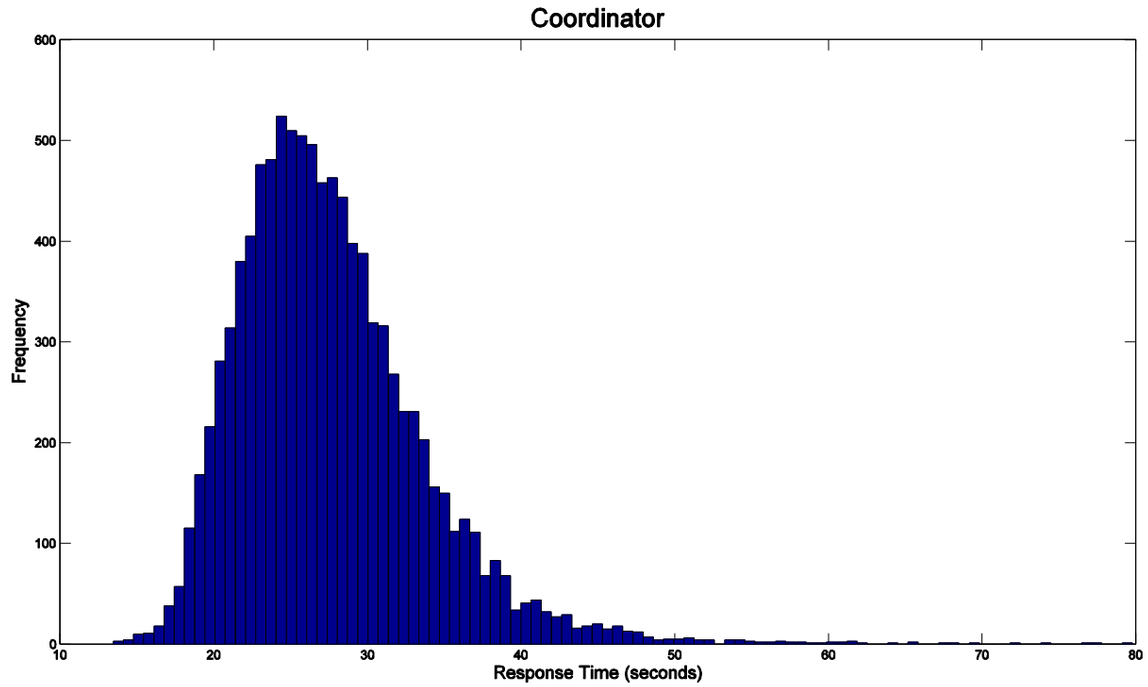
What is the expected response time?

Large number of possible configurations



QoS (response time) of Atomic Services in the Composite Service



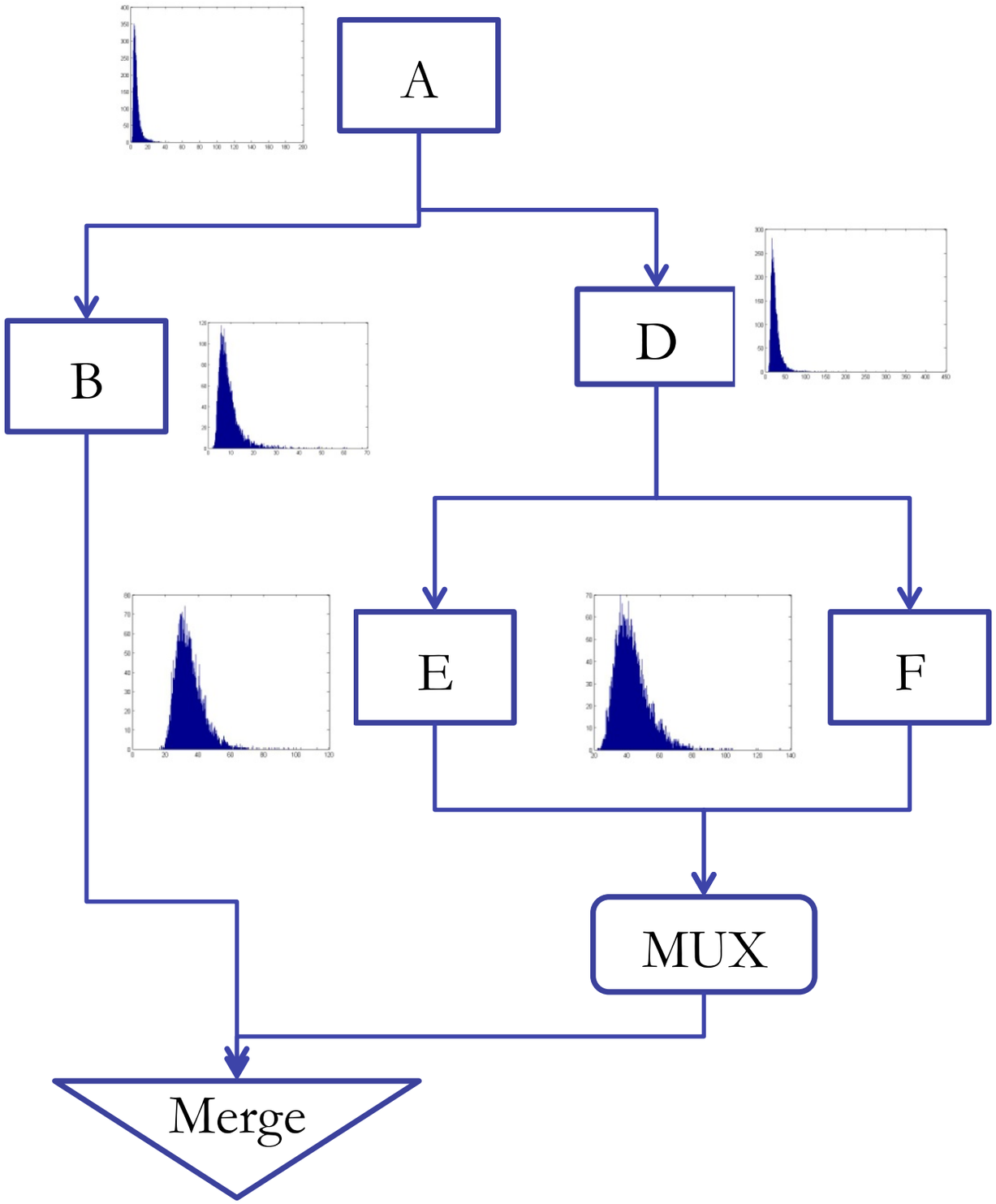


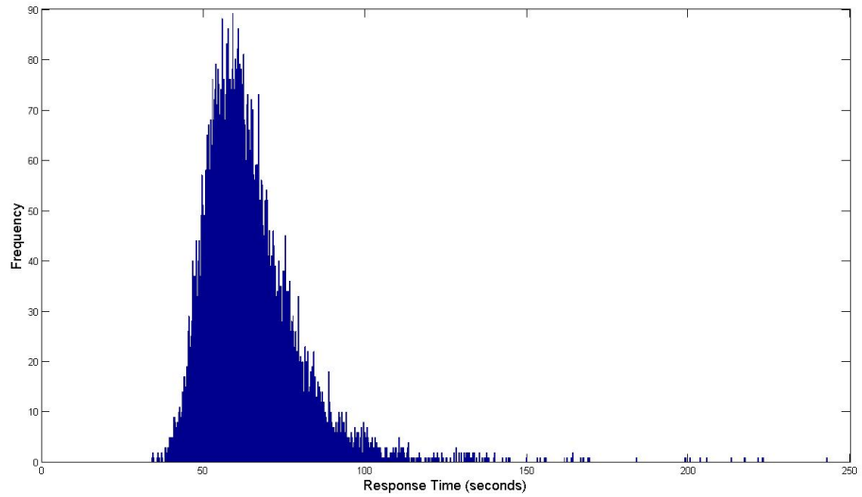
First Aid Material

Co-ordinator

Soft contract: probability distribution of QoS for each atomic service

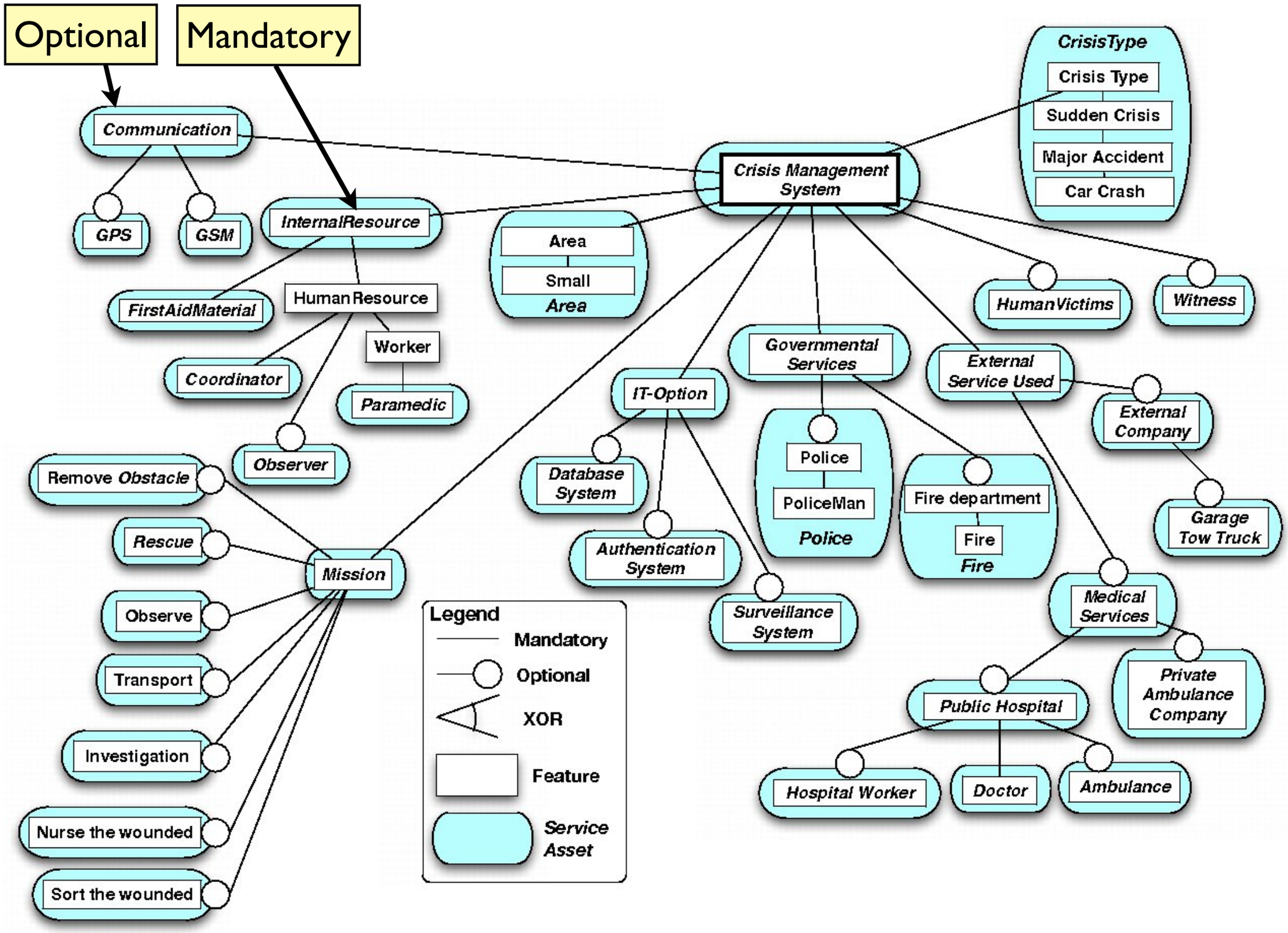
Composite service QoS for ONE configuration





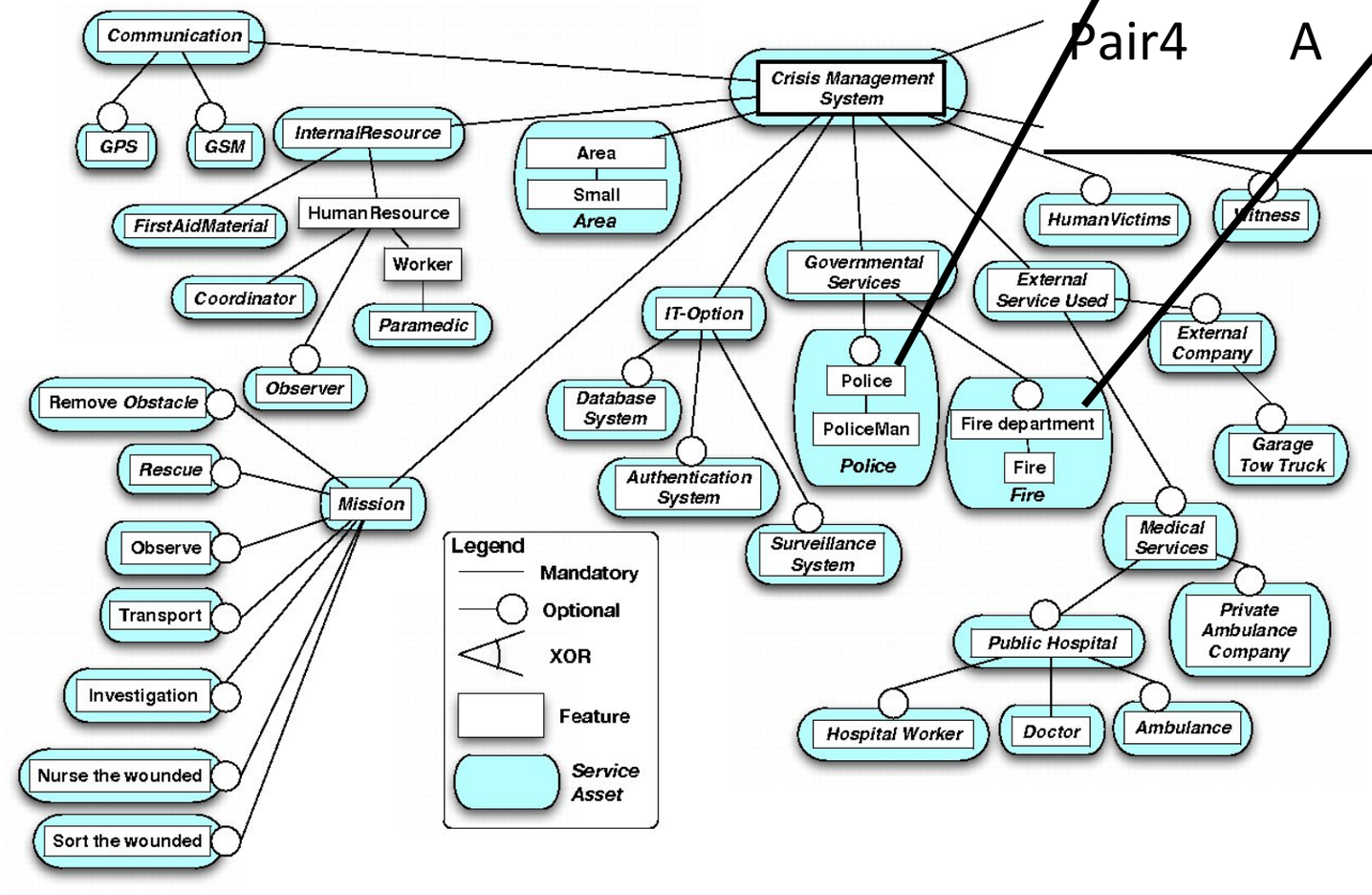
Composite Service response time distribution

Feature Model in Car Crash Crisis Management System

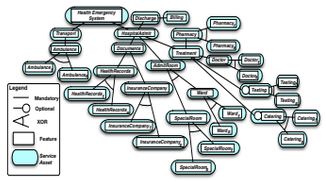


C(41,2) = 820 Possible Pairwise Interaction

	Police	Fire Dept.
Pair1	N/A	N/A
Pair2	N/A	A
Pair3	A	N/A
Pair4	A	A



Constraint Solving to Generate Configurations Covering T-wise



Feature Diagram

FD

1. FD2Alloy

T-wise Strategy

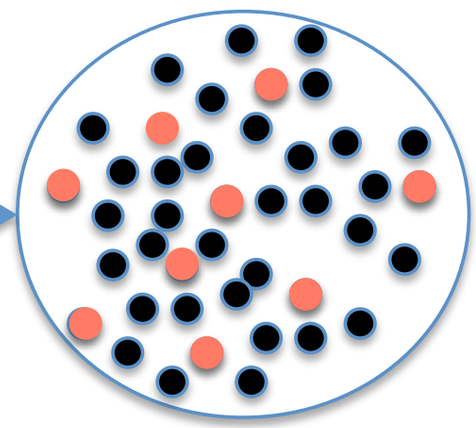
T (Eg. T=2 implies pairwise)

2. Generation of Alloy Predicates for Tuples (pairs for T=2)

Alloy Model

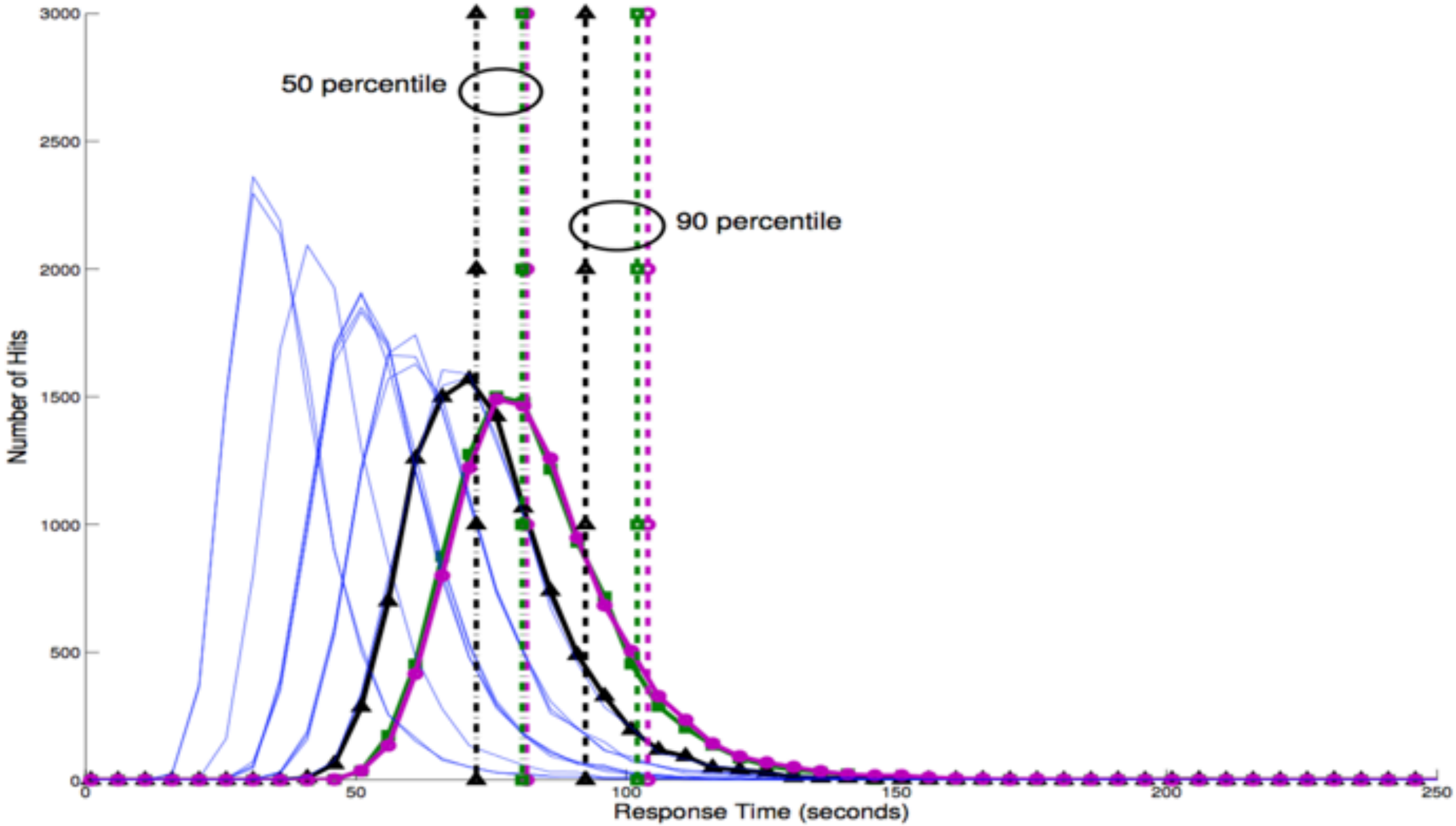
3. Detection of Consistent Tuples (w.r.t FD)

4. Generate Configurations covering valid pairs (Divide and Compose)

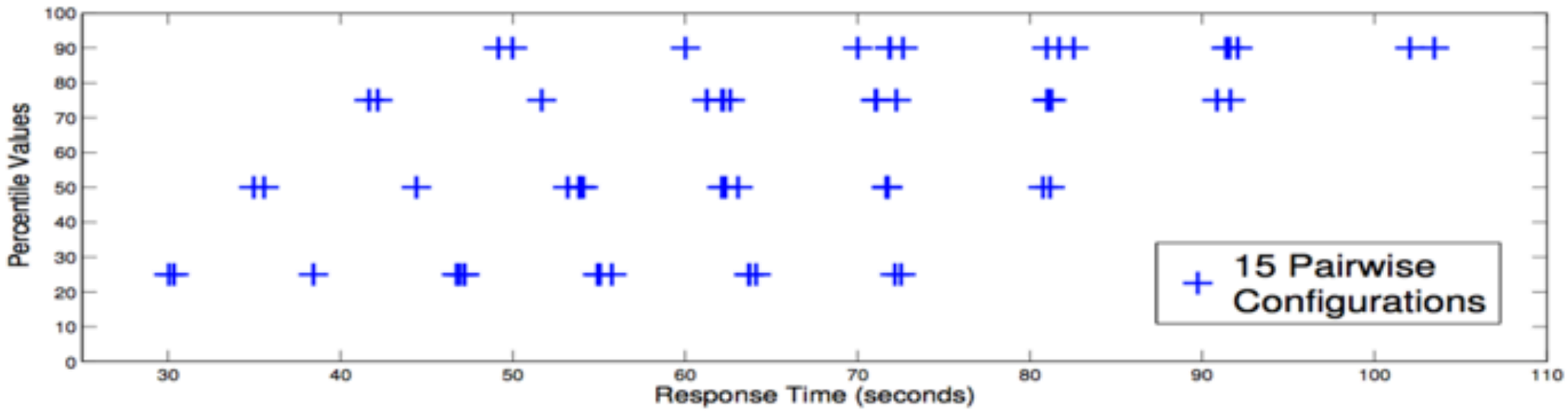


We discover only 15 configurations covering all valid pairs in the 820 feature interaction pairs

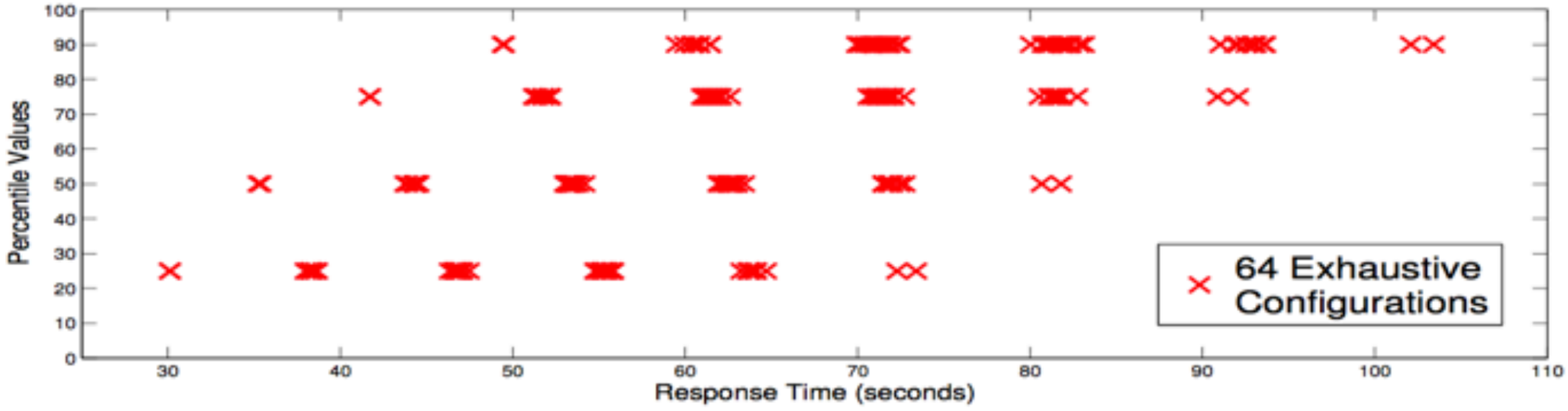
Variability in QoS (response time)



Coverage of Pairwise vs. All Configurations



(a)

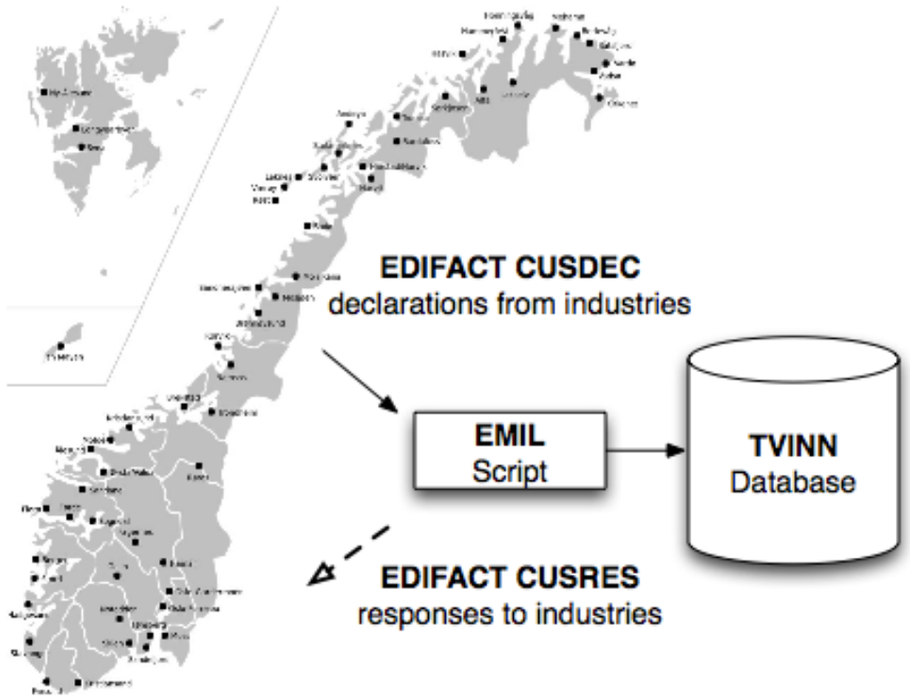


(b)

Outline

- Introduction
- Case 1: Tests that Cover All T-wise Interactions
- Case 2: Finding Service Level Agreements
- **Case 3: Interaction Coverage in Data-intensive Systems**
- Case 4: Dynamically Adaptive Vision Systems
- Case 5: Model Checking for Test Generation
- Conclusion

The Heart of Norway's E-governance: TVINN

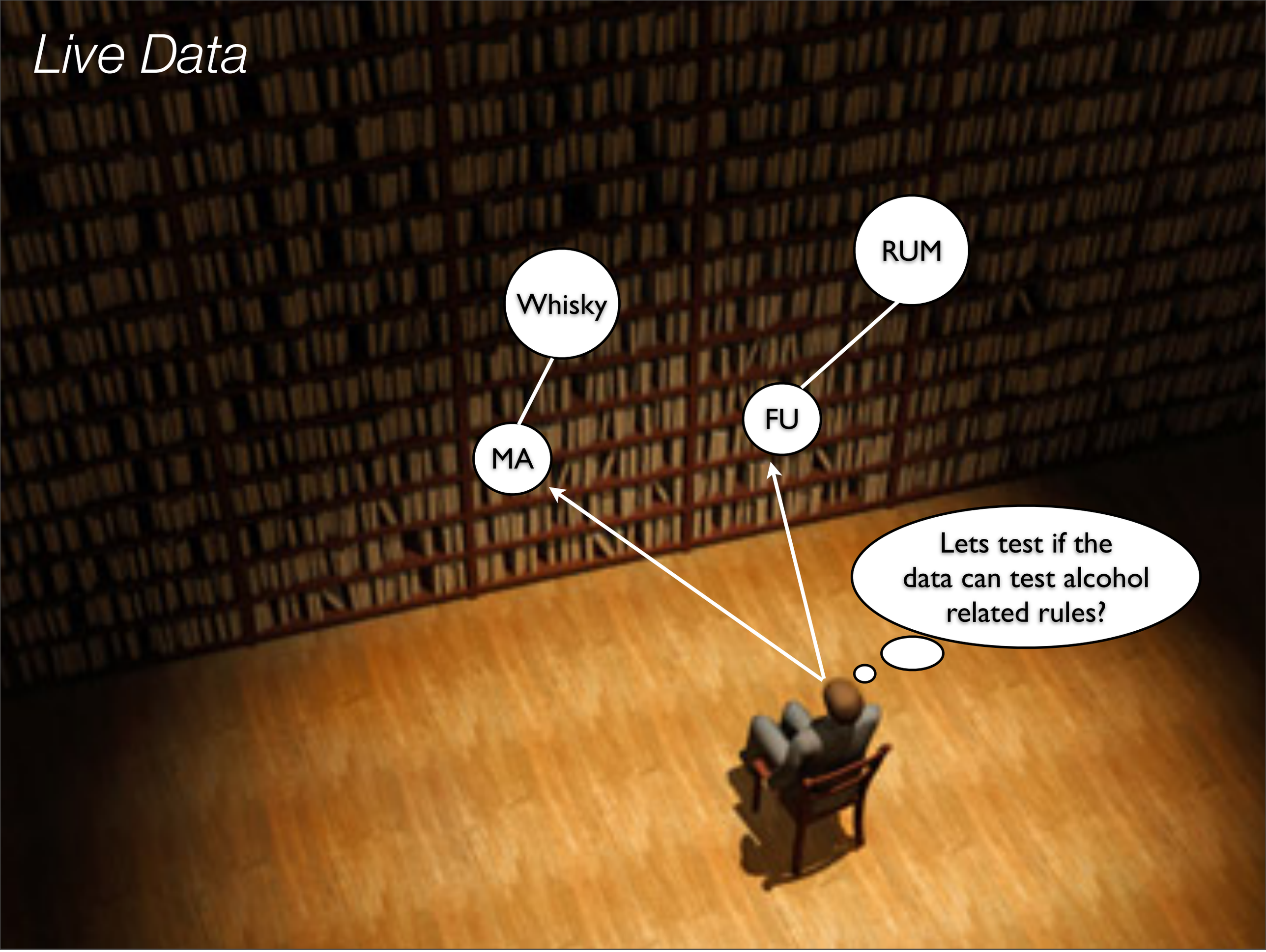


Norwegian Toll Customs uses the **TVINN** system to handle about 30,000 declarations/day

- 8000 to 30,000 declarations/day, potentially adhering to about 200,000 customs rules
- Customs rules typically **accept/return** declarations based on information in the declaration
- Towards a **corruption-free and efficient information society**

Live Data

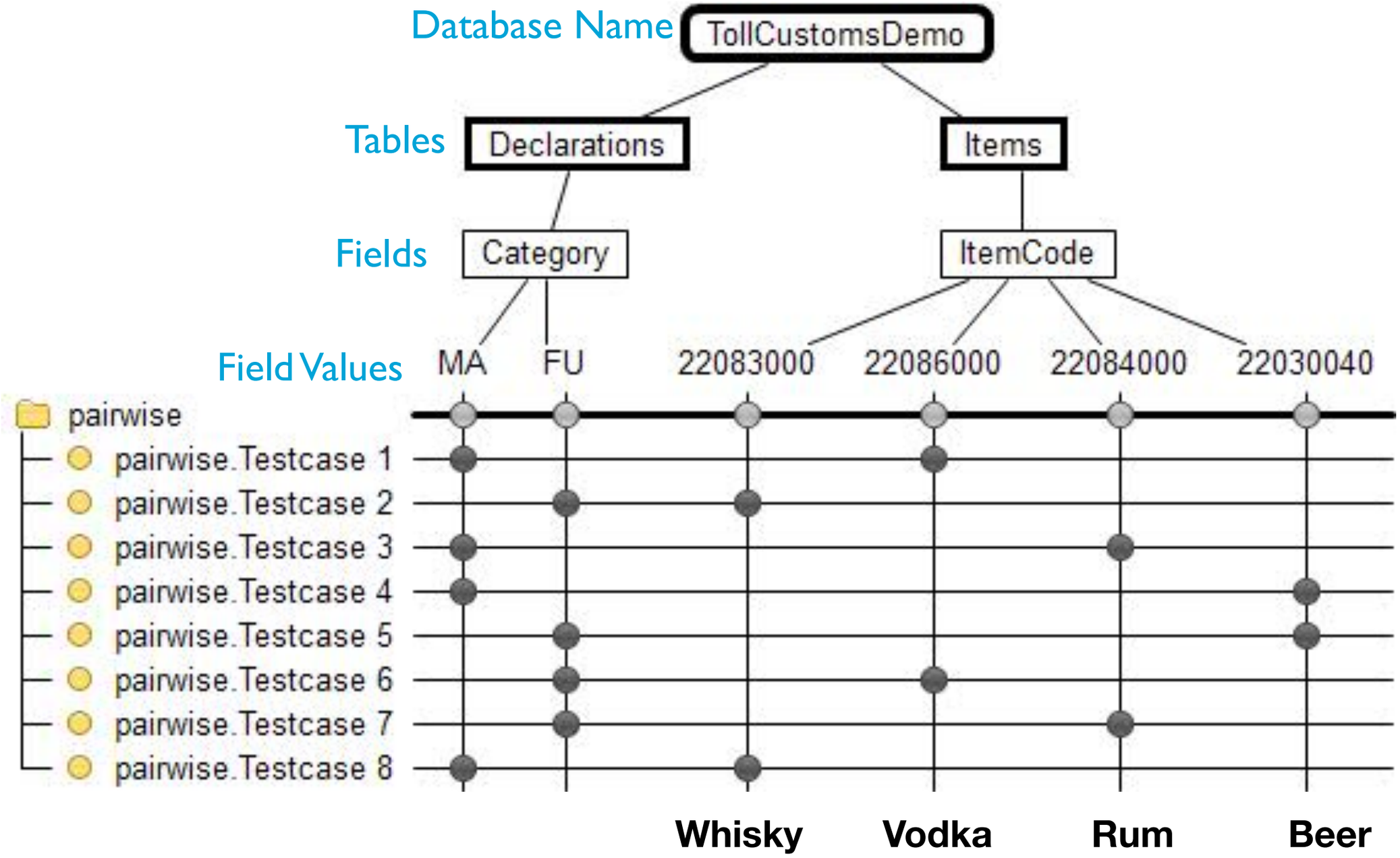
Live Data


RUM
Whisky
FU
MA
Lets test if the data can test alcohol related rules?

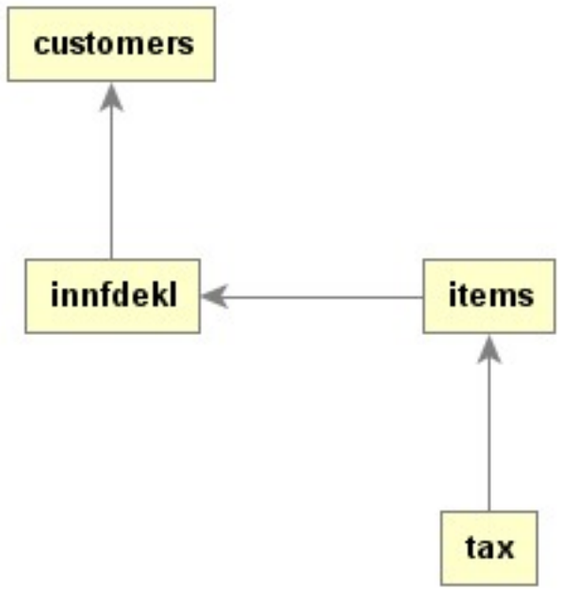
Customs business rules are a combination of
10,000 Item Codes, **88 Country Groups**, **934 Tax Types, 5 Declaration Categories**

Trillions of possible ways they interact! Only about **200,000 rules** used in practice.
What are the rules tested by my live data?

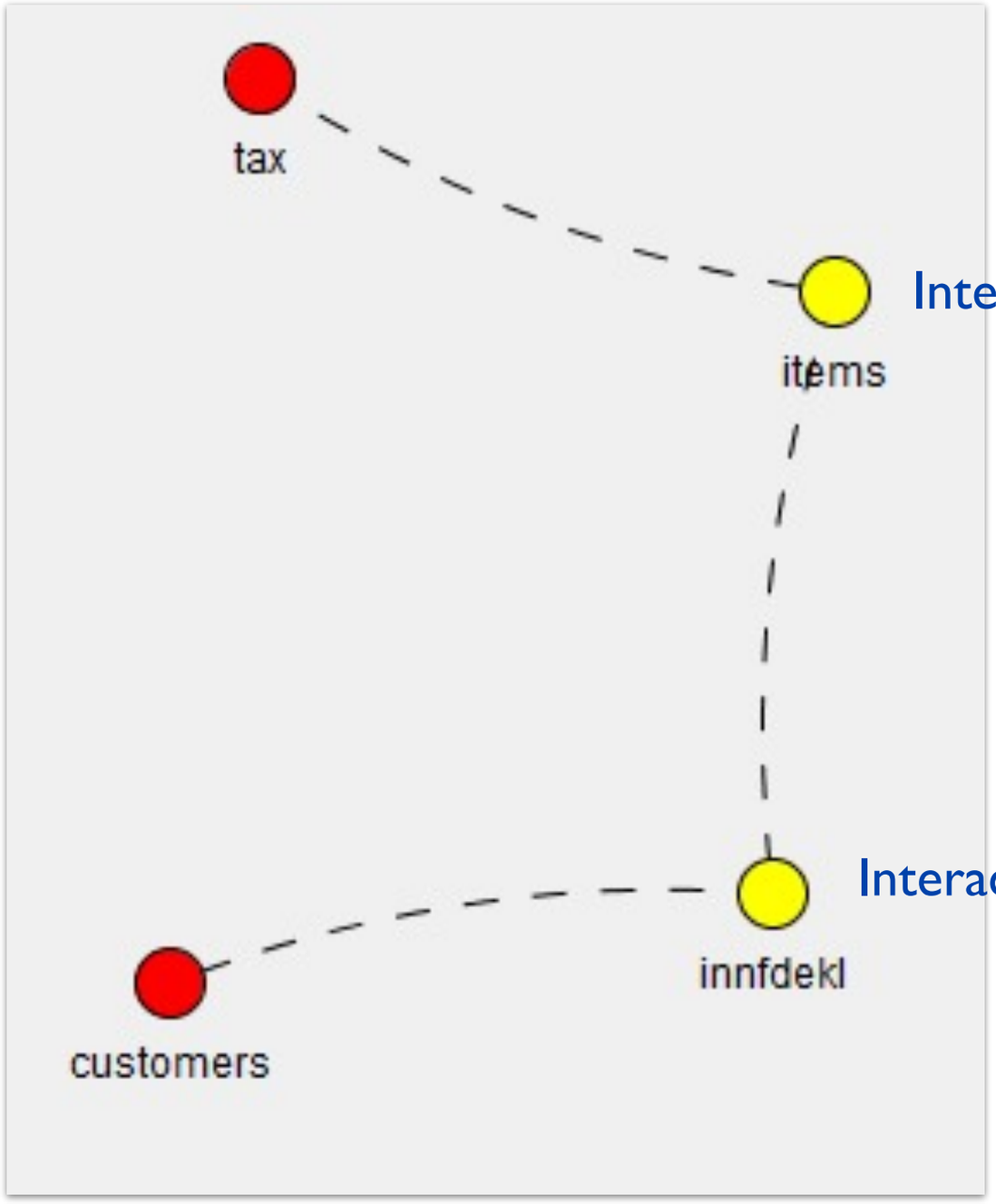
Modelling Data Interactions



Extracted Graph from Database Schema



Database Schema at Toll

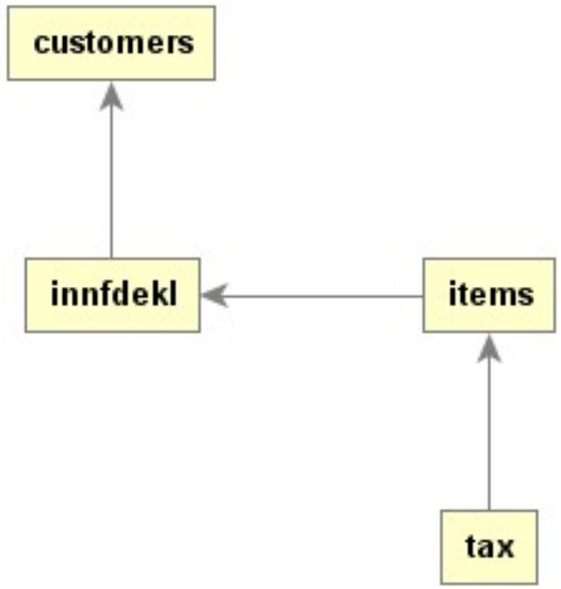


Interacting Table 1

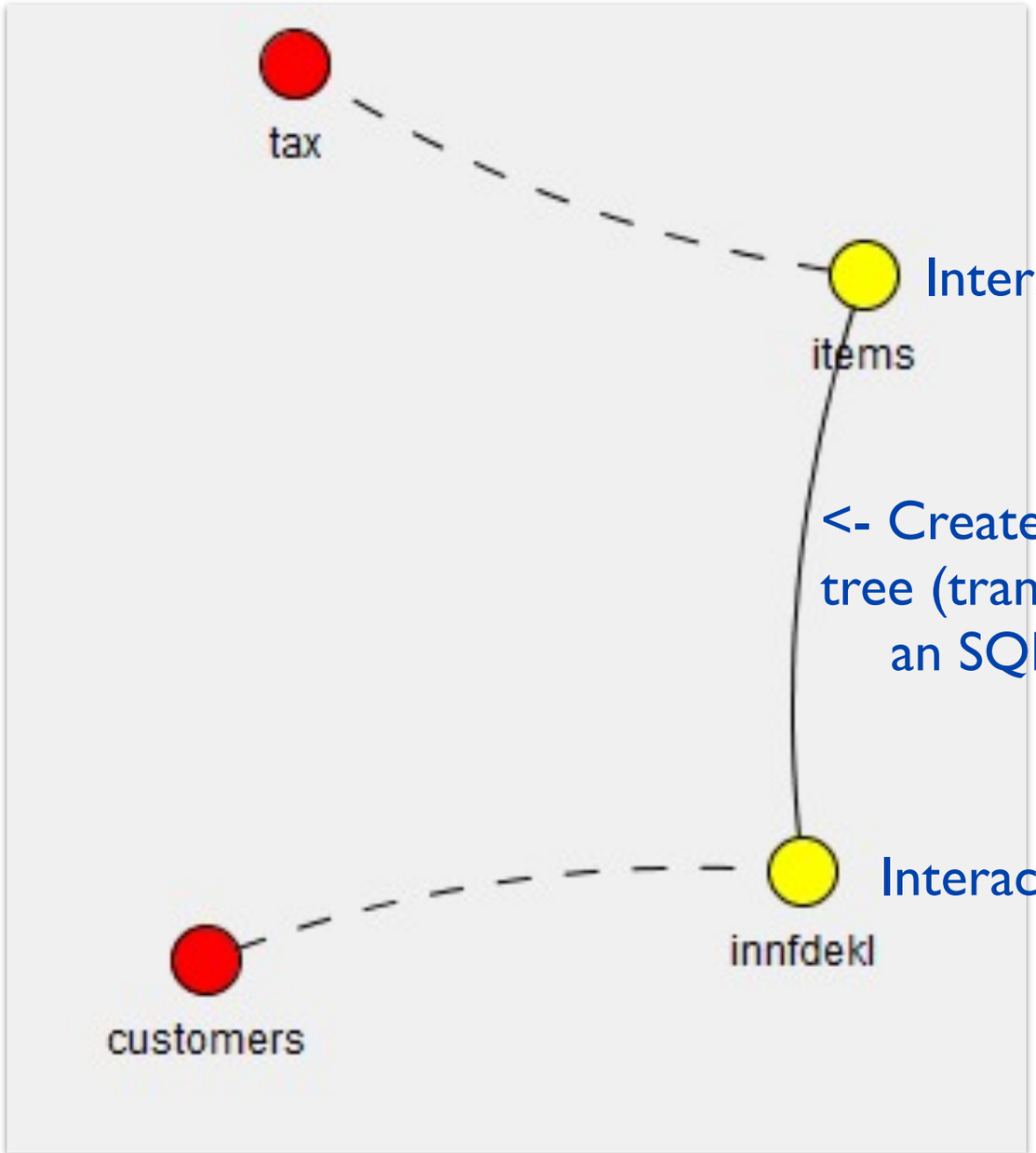
Interacting Table 2

Graph of the Schema

Extracted Graph from Database Schema

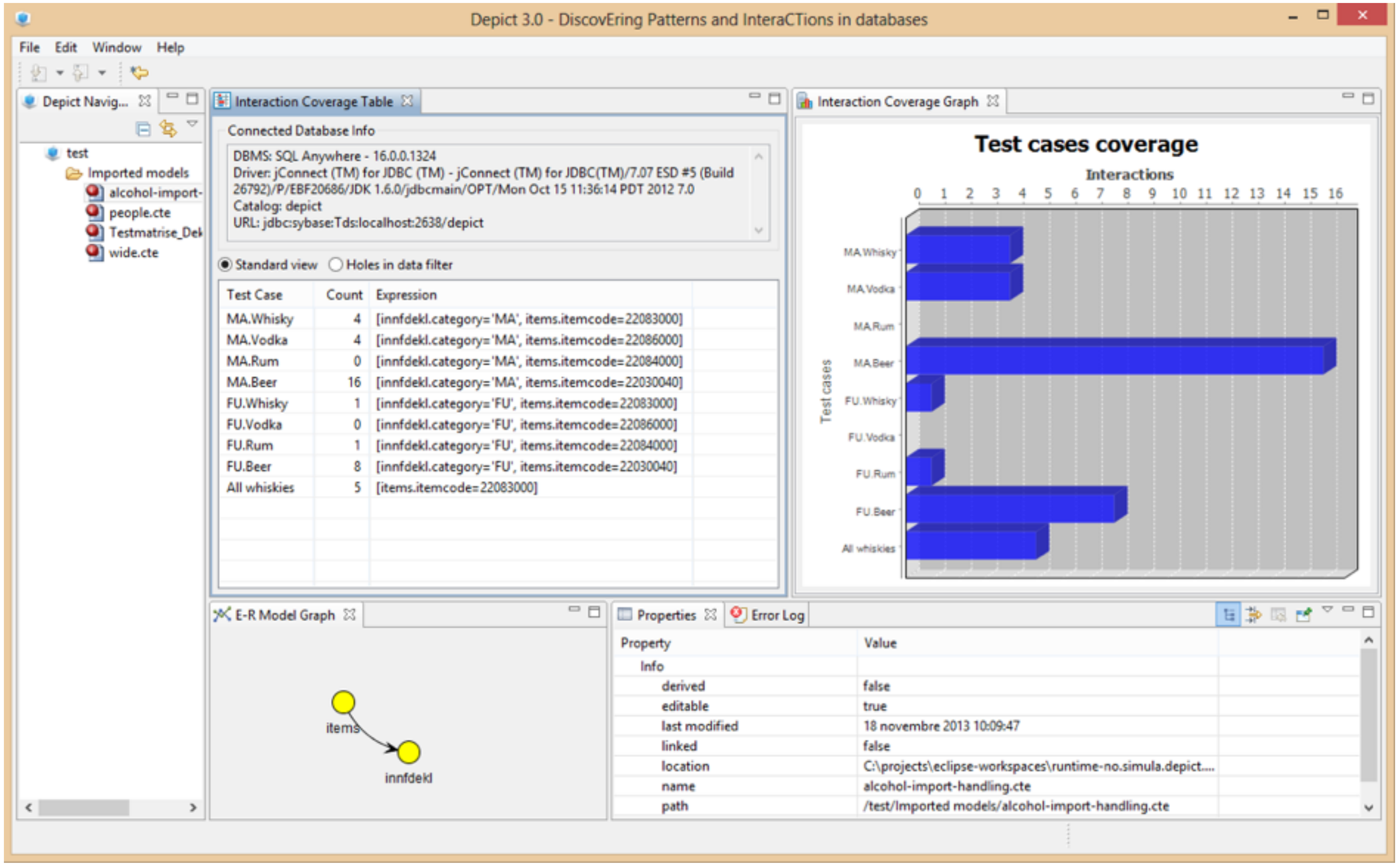


Database Schema at Toll



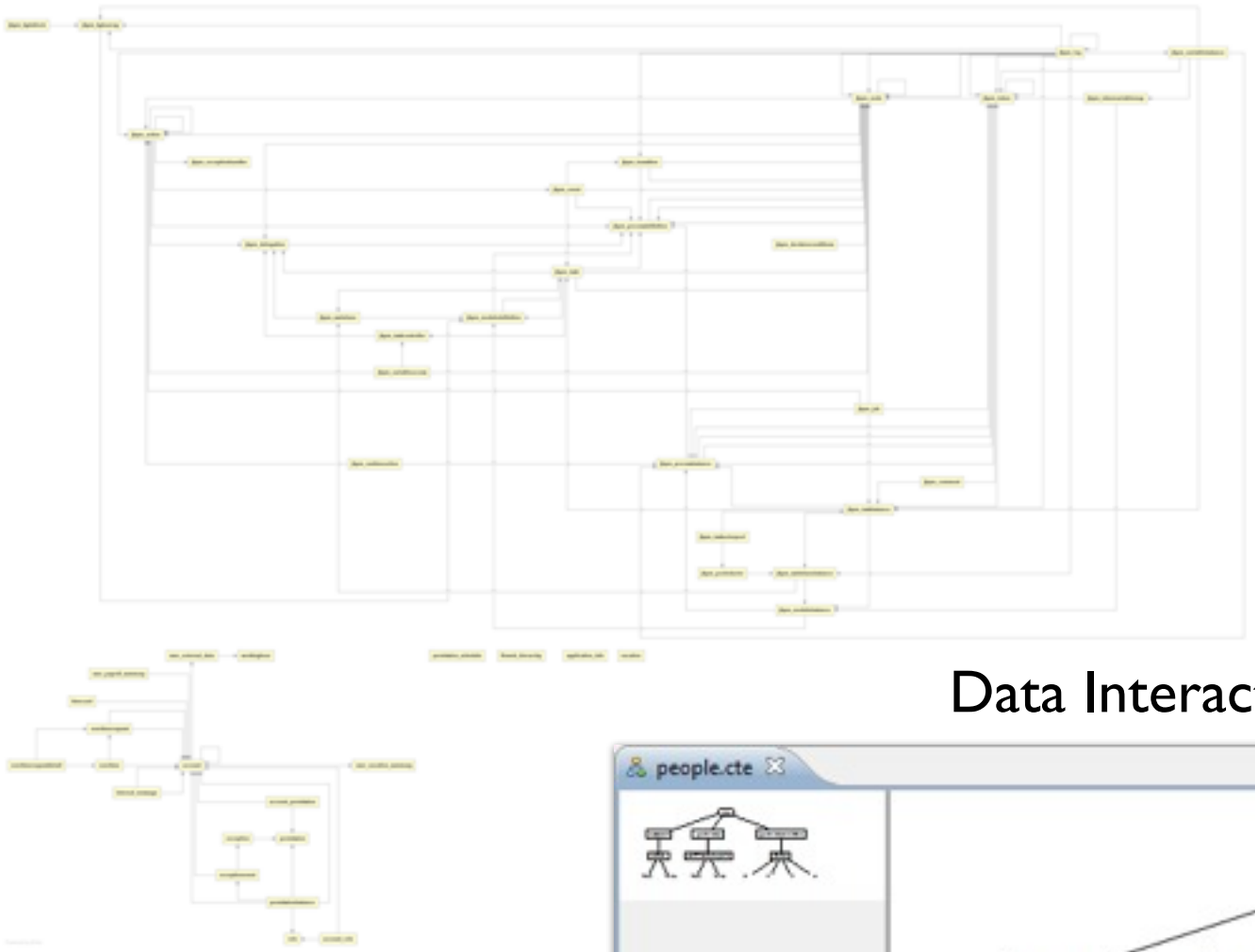
Graph of the Schema

Query Generation and Interaction Coverage Analysis

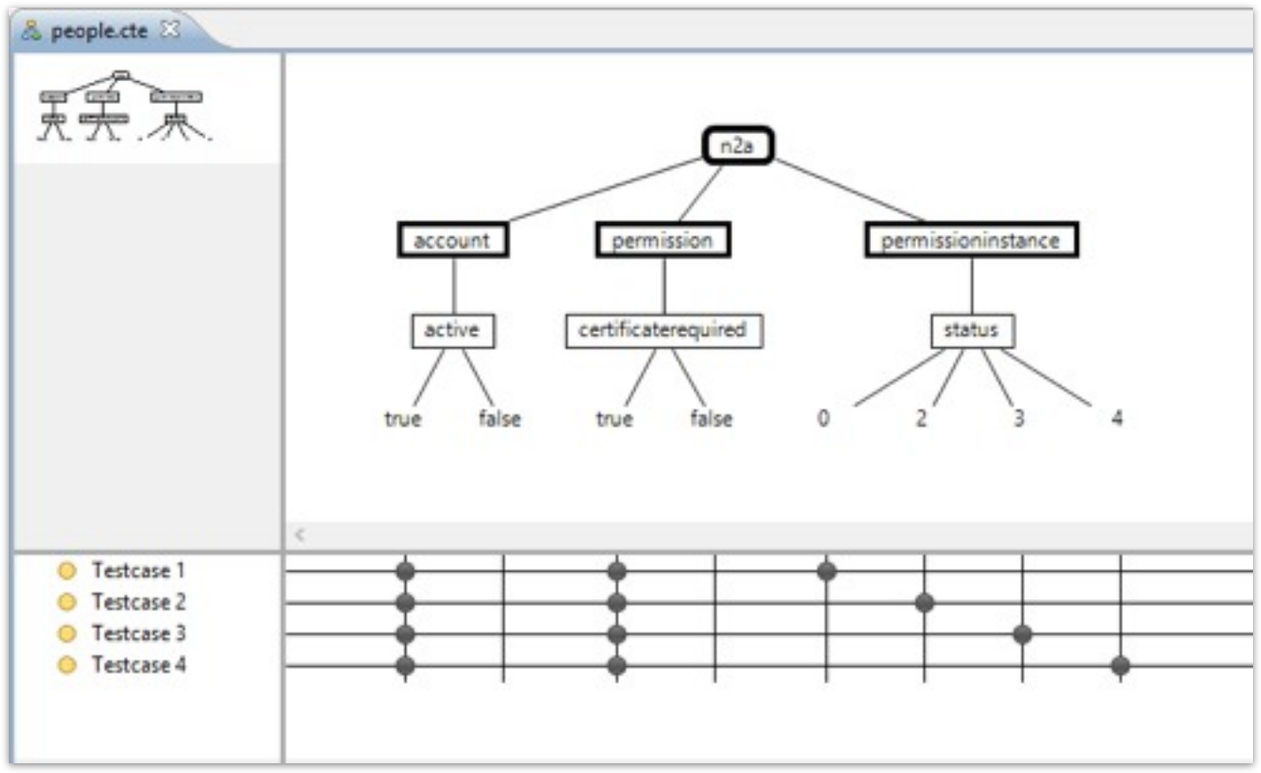


Real Industrial Example: ACME Chemicals

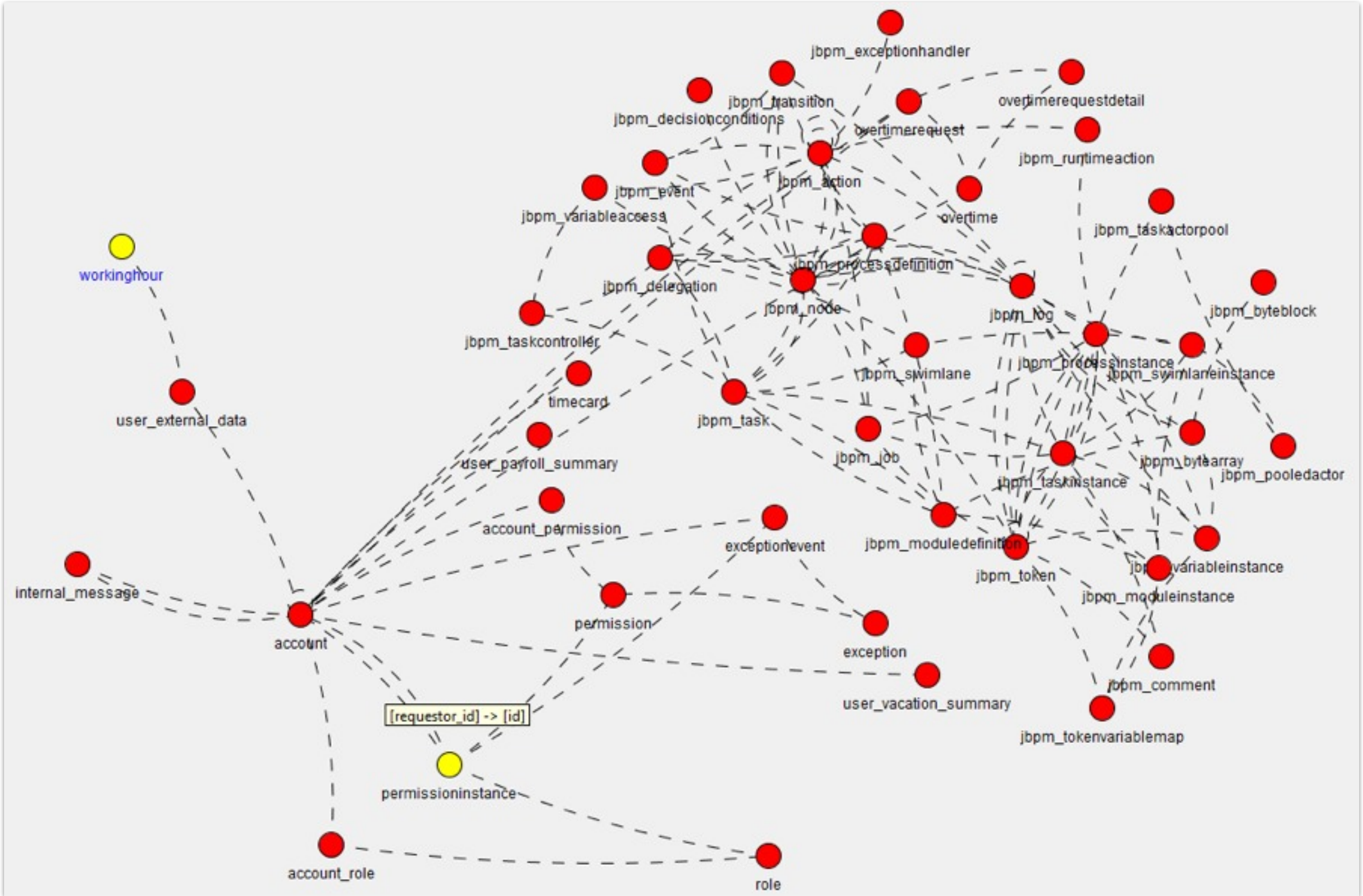
Database schema



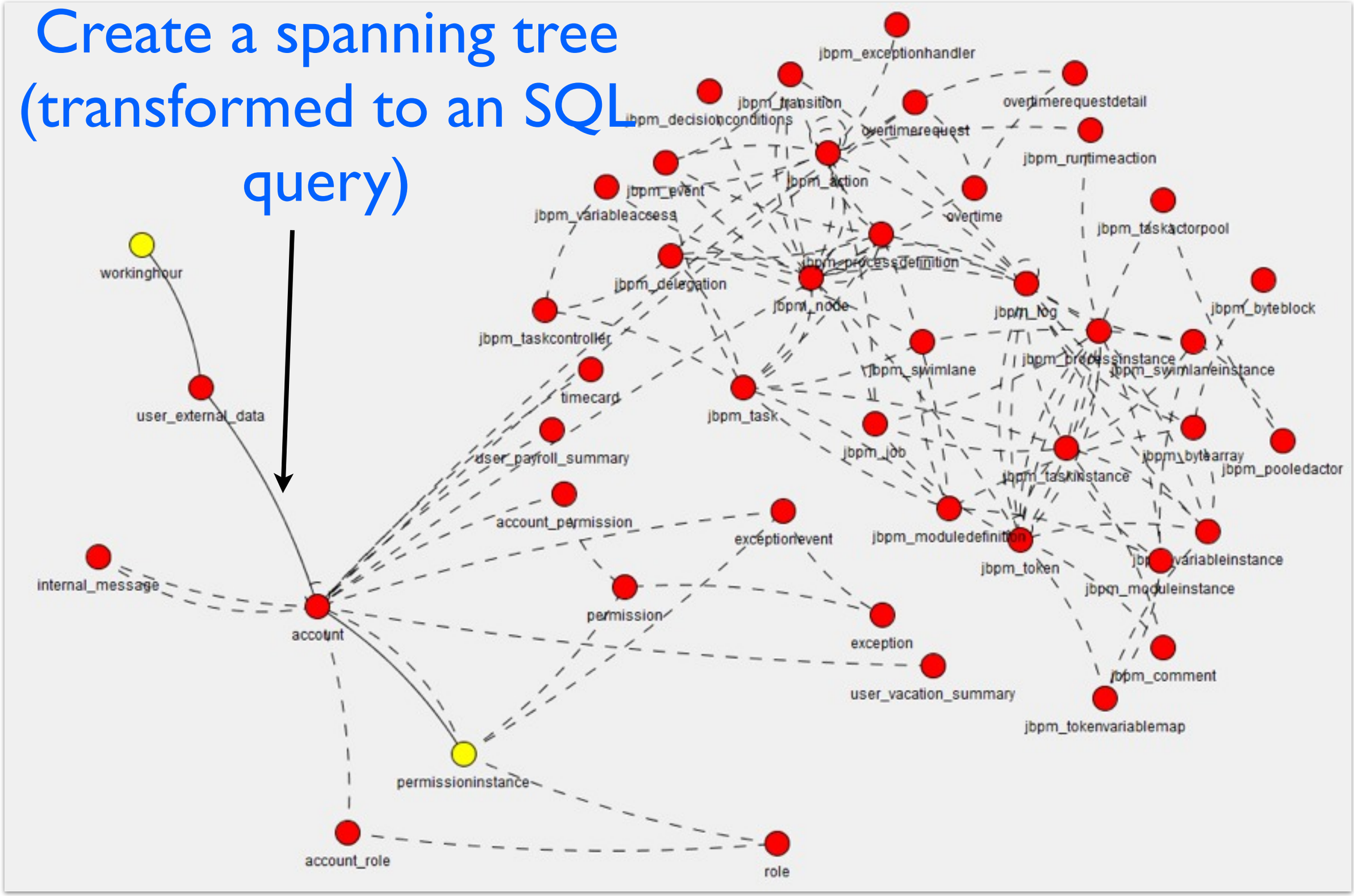
Data Interaction Model



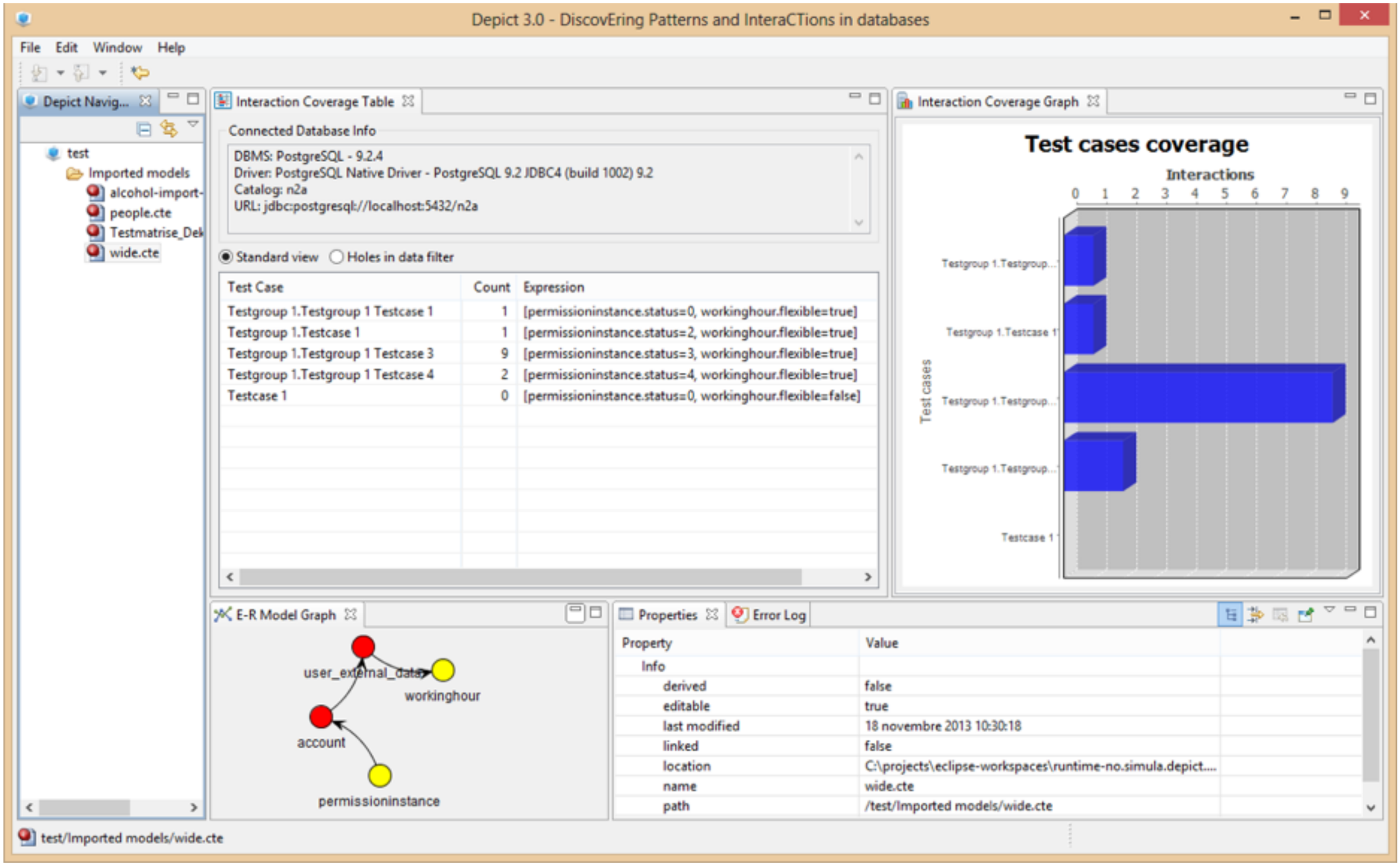
Real Industrial Example: ACME Chemicals



Real Industrial Example: ACME Chemicals



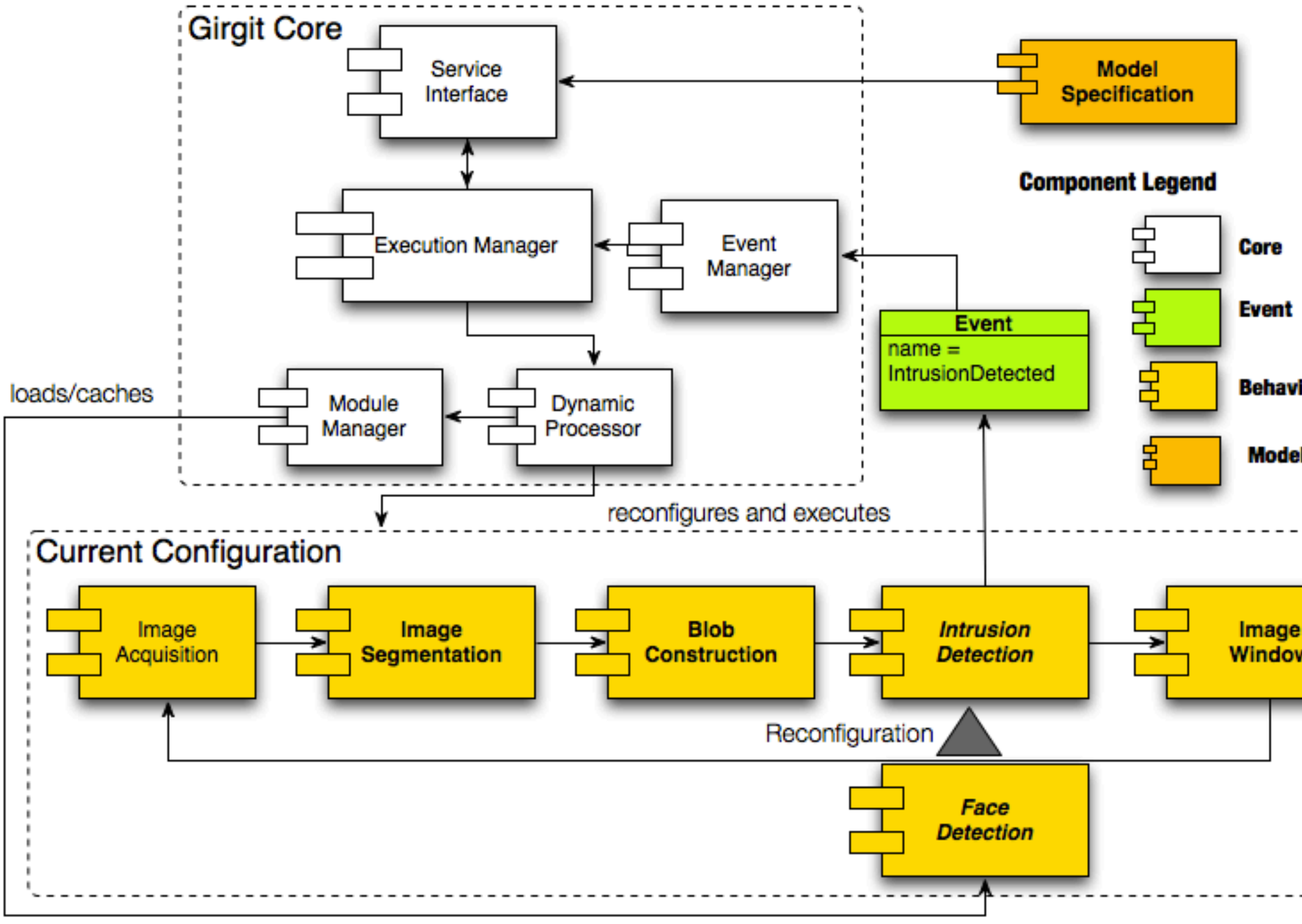
Query Generation and Interaction Coverage Analysis



Outline

- Introduction
- Case 1: Tests that Cover All T-wise Interactions
- Case 2: Finding Service Level Agreements
- Case 3: Interaction Coverage in Data-intensive Systems
- **Case 4: Dynamically Adaptive Vision Systems**
- Case 5: Model Checking for Test Generation
- Conclusion

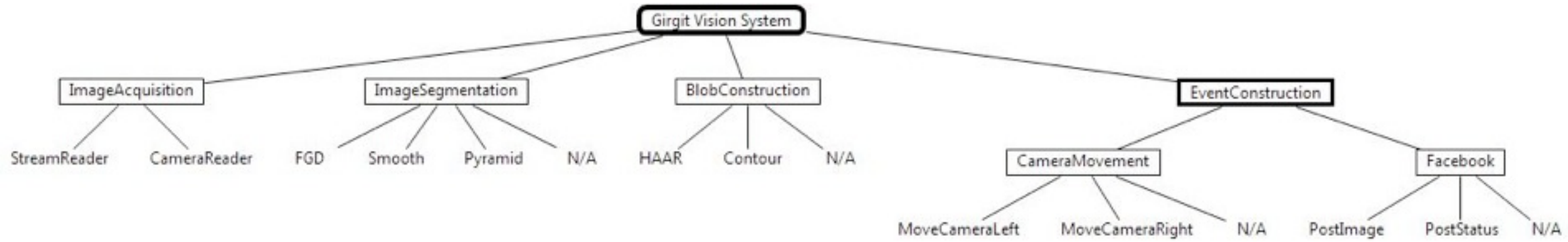
Case 4 : Girgit, A dynamically adaptive vision system



Case 4 : Girgit, A dynamically adaptive vision system

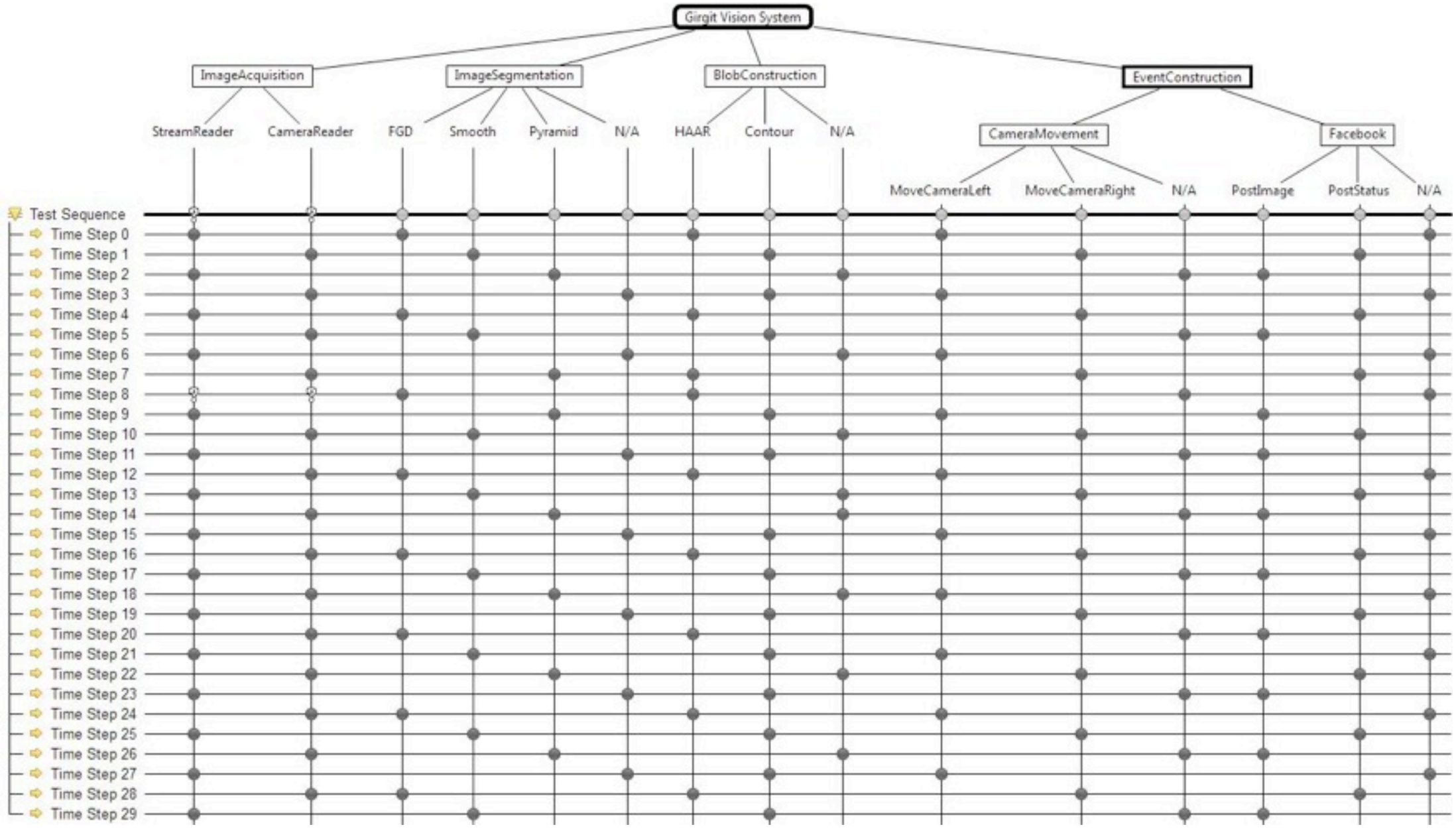
Adaptation from Intrusion Detection to Face Detection

Case 4 : Girgit, A dynamically adaptive vision system



Modelling Vision System Variability

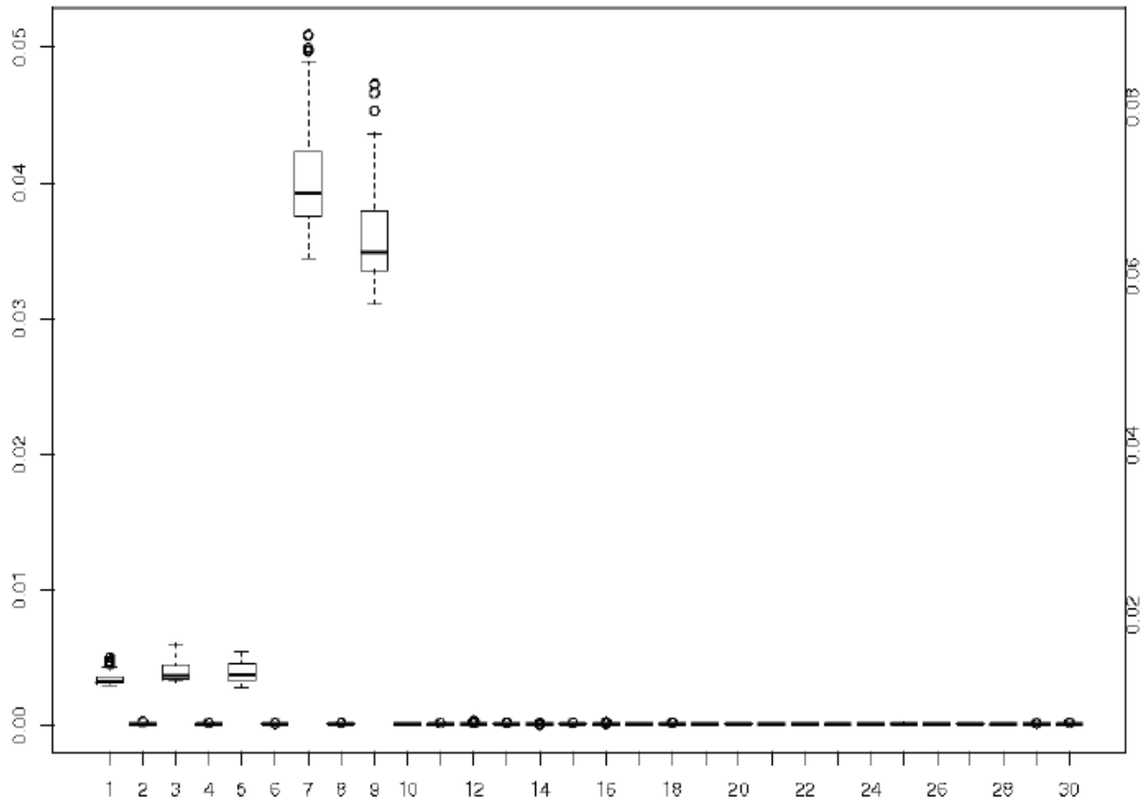
Case 4 : Girgit, A dynamically adaptive vision system



Modelling a Test Sequence in Time

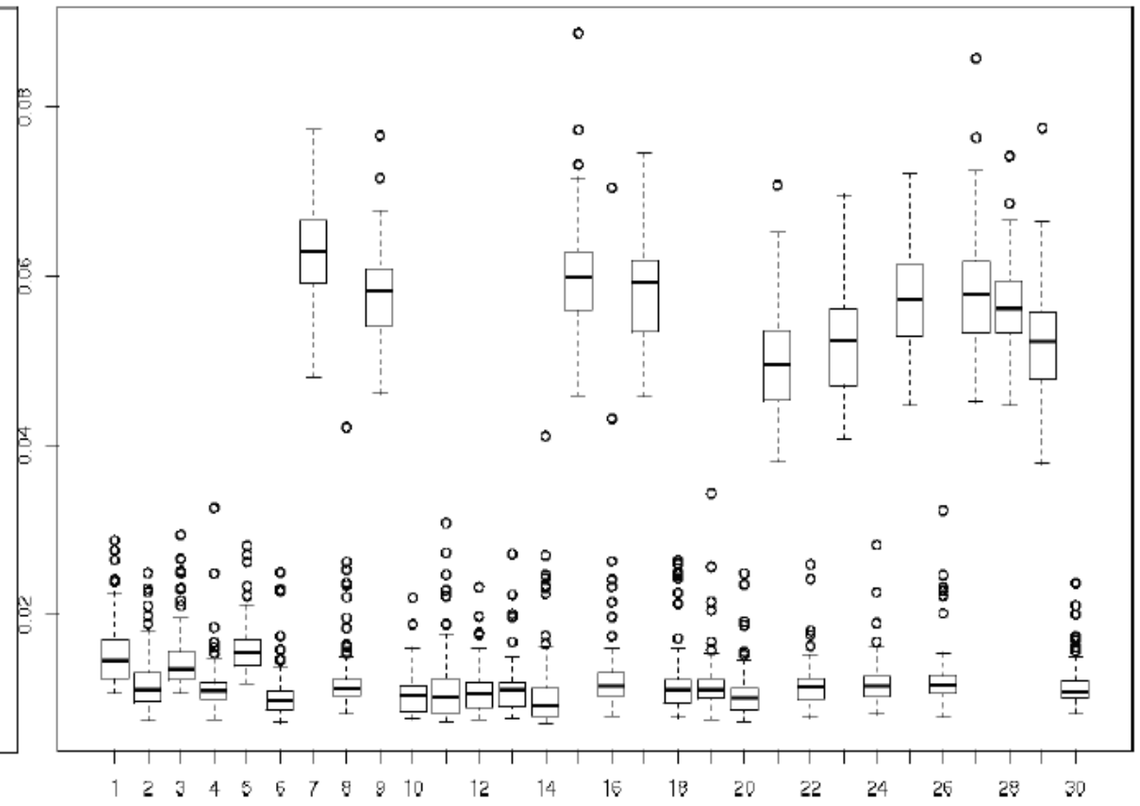
Case 4 : Girgit, A dynamically adaptive vision system

A new dimension of variability: time



(a) Caching ON

Average 10 mu secs



(b) Caching OFF

Average 2 micro secs

Adaptation Time for 30 Reconfigurations
between 8 possible configurations

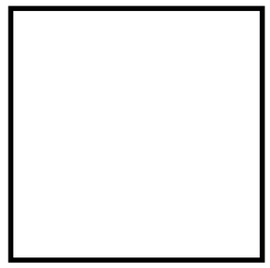
Outline

- Introduction
- Case 1: Tests that Cover All T-wise Interactions
- Case 2: Finding Service Level Agreements
- Case 3: Interaction Coverage in Data-intensive Systems
- Case 4: Dynamically Adaptive Vision Systems
- **Case 5: Model Checking for Test Generation**
- Conclusion

Test cases

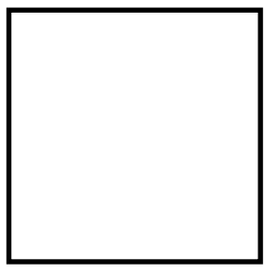
Set of (input,output) pairs which we don't often have

What we have?



Software system coded by developers (white-box)

If we can represent a system as a formal model



Formal model in Alloy

Alloy is a lightweight formal method that represents system models as a *set of relations* and can generate examples and counterexamples via SAT solving

For example Alloy is a lightweight formal method

```
module modelCheck
open util/boolean as Bool
one sig System
{
input: one Bool,
feature1: one Bool,
feature2: one Bool,
feature3: one Bool,
output : one Bool
}
fact SystemComputation
{
output =  input & feature1 + feature2 & feature3
}
```

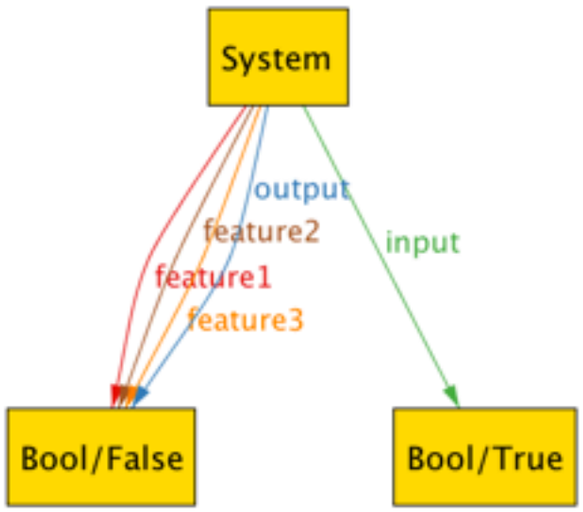
A fact about a boolean computation in the system

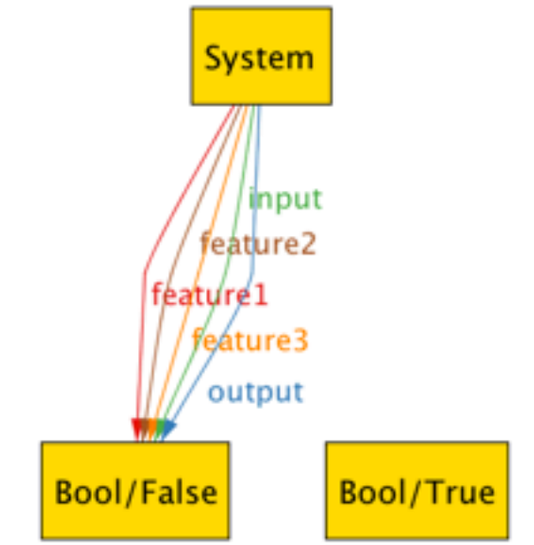
Configuring the features of a system and example generation

```
module modelCheck
open util/boolean as Bool
one sig System
{
input: one Bool,
feature1: one Bool,
feature2: one Bool,
feature3: one Bool,
output : one Bool
}
fact SystemComputation
{
output =  input & feature1  + feature2 & feature3
}
pred configuration1_example
{
System.feature1=False and System.feature2=False
and System.feature3=False
}
run configuration1_example for  1
```

Generated Instances and Derivation of Test Cases

Scenarios where the system configuration always holds





Test Case 1

(input = true, output=false)

Test Case 2

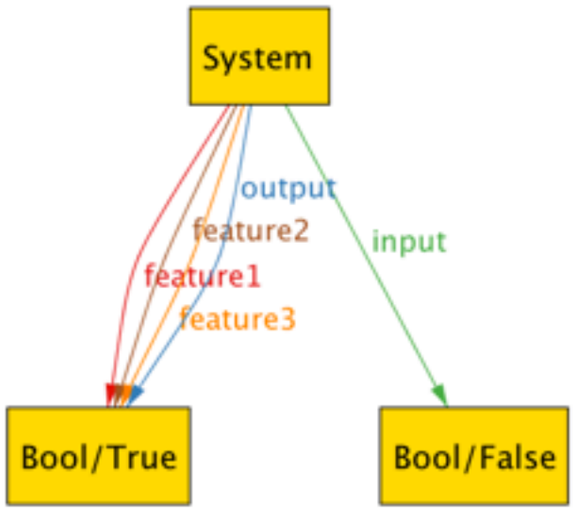
(input = false, output=false)

Configuring the features of a system and counterexample generation

```
module modelCheck
open util/boolean as Bool
one sig System
{
input: one Bool,
feature1: one Bool,
feature2: one Bool,
feature3: one Bool,
output : one Bool
}
fact SystemComputation
{
output =  input & feature1  + feature2 & feature3
}
assert configuration1_counterExample
{
System.feature1=False and System.feature2=False
and System.feature3=False
}
check configuration1_counterExample for  1
```

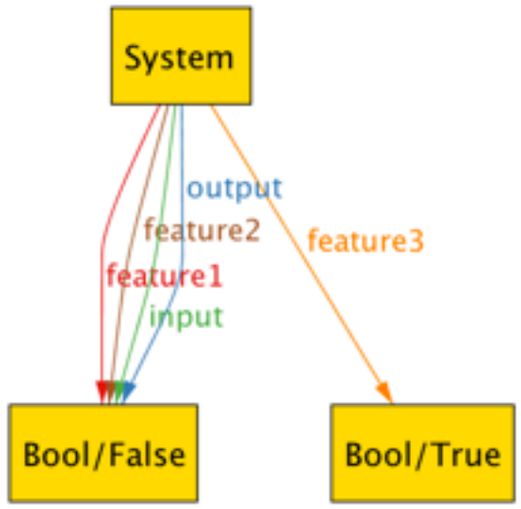
Generated Counterexamples and Derivation of Test Cases

Scenarios where the configuration does not hold



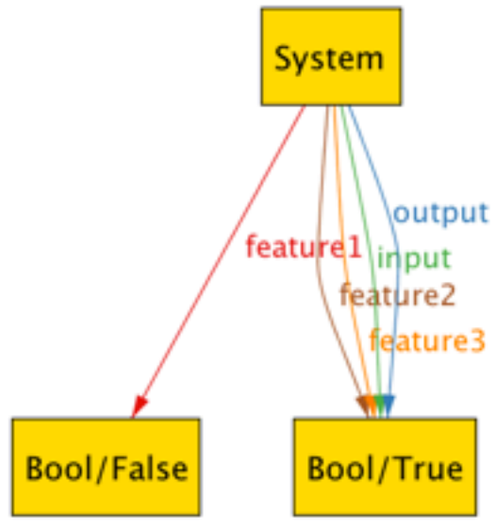
Test Case 1

(input = false, output=true)



Test Case 2

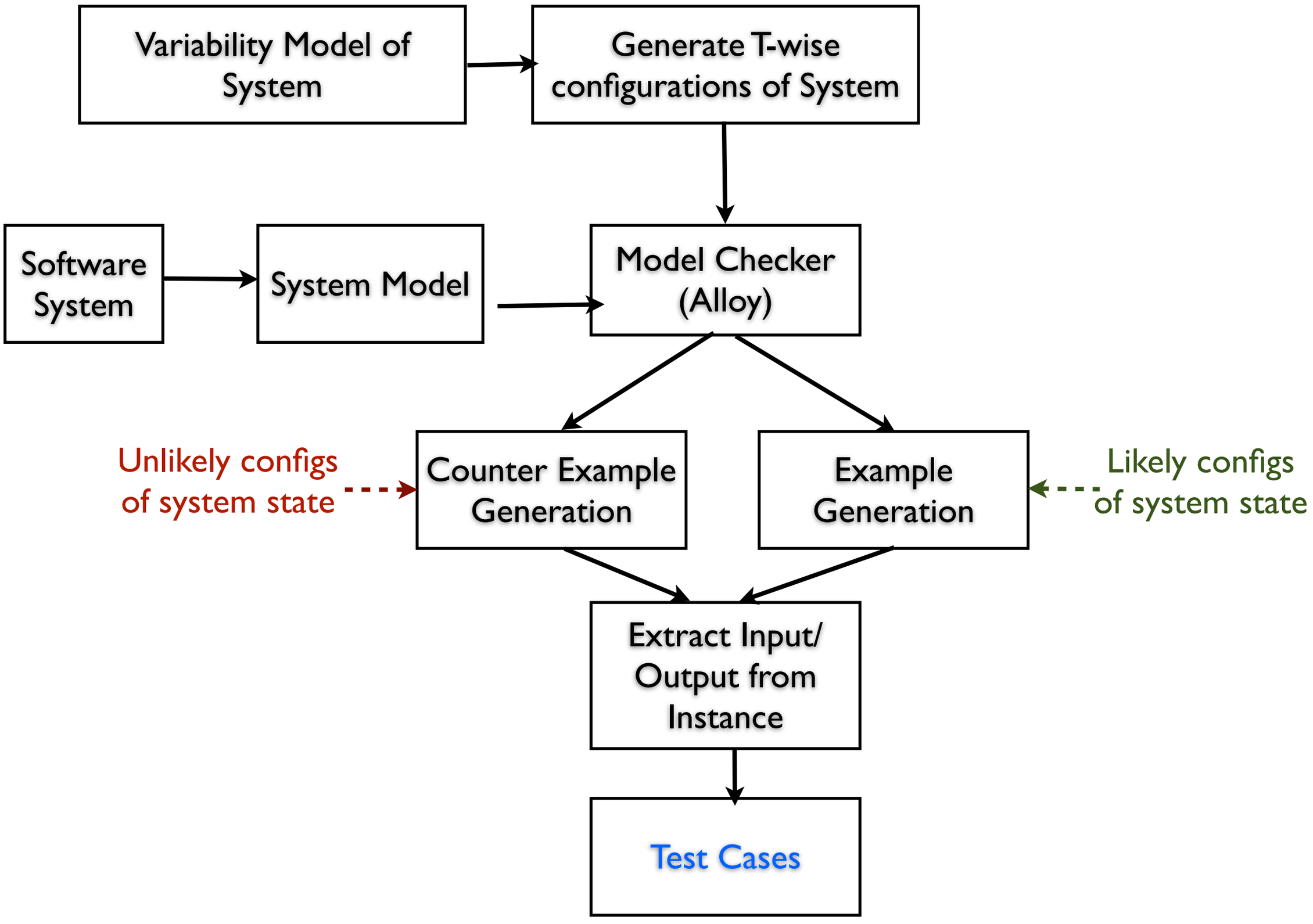
(input = false, output=false)



Test Case 3

(input = true, output=true)

Overall Process



Outline

- Introduction
- Case 1: Tests that Cover All T-wise Interactions
- Case 2: Finding Service Level Agreements
- Case 3: Interaction Coverage in Data-intensive Systems
- Case 4: Dynamically Adaptive Vision Systems
- Case 5: Model Checking for Test Generation
- **Conclusion**

Conclusion

- Combinatorial interaction testing greatly reduces the number of configurations
- Effective coverage of the software configuration space (Car Crisis Management)
- Interaction coverage is effective in data-intensive systems (Toll Customs)
- One can also imagine interaction coverage between configurations in a sequence to discover faults in QoS (Self-adaptive vision system)
- System configurations covering T-wise interactions combined with model checking can help automatically generate test cases

Thank you.