

# QONCEPT

## Semantic Query Optimisation in Complex Event Processing Technologies

François Bry and Olga Poppe

Institute for Informatics, University of Munich, Germany

February 5, 2010

## Project Details in Brief

<b>Project Acronym</b>	QONCEPT
<b>Action Line</b>	Complex Event Processing, Databases, Knowledge presentation
<b>Funding</b>	German Research Foundation (DFG)
<b>Participants</b>	University of Munich
<b>Start Date</b>	2009-11-01
<b>End Date</b>	2011-10-31 (2012-10-31)
<b>Duration</b>	2 years (3 years)
<b>www</b>	<a href="http://www.pms.ifi.lmu.de/qoncept/">http://www.pms.ifi.lmu.de/qoncept/</a>

# Connection to EMILI

## Semantic query optimisation



WP4: LMU WP5: CWI

QONCEPT



WP3: SKYTEC, PUPIN, AIA, ASIT



## Real-life applications in CEP

# The Goal of QONCEPT

Semantic

Query Optimisation in CEP Technologies

By means of  
application-  
specific  
knowledge

Efficiency of query evaluation  
Simplicity of queries

For arbitrary Event  
Query Languages  
(EQLs)

# Application-specific Knowledge

## Baggage Sorting System

- Each bag is in one place at each point of time
- It takes long to transport a bag over a long distance

## Sales

- For each buy there are three actions to be done: order, shipping and tracking. Each action exactly once and in this order
- Each item can be sold only to one person, but the same person can buy many items

## Fire Detection

- Sensors which are located in the same room deliver similar measurements within a short period of time
- Only combined measurements of different near located sensors within a short period of time can indicate fire

# Application-specific Knowledge

## Online Auction

- Many auctions may take place simultaneously
- The first 20 min of each auction are for the bidder enrolment
- A bidder may take part in an auction only if he has enrolled for this auction
- In an auction without bidders, no items are presented
- After the bidder enrolment at least one item is presented in each auction
- A bid for an item must offer a higher price than all earlier bids for the item
- After 30 sec without bids, a hammer beat comes
- There are 3 hammer beats. After the 1<sup>st</sup> and the 2<sup>nd</sup> ones new bids may come. After the 3<sup>rd</sup> one, the item is sold
- If there is no bid for an item, the item will not be sold
- An item is presented after the bidder enrolment or after a previous item has been considered within the same auction

## Events for an Online Auction

```
auction {  
  auctionID [ 123 ],  
  category [ furniture ],  
  currency [ euro ],  
  sellerID [ 456 ]  
} t=[1,11]
```

```
bidderEnrolment {  
  auctionID [ 123 ],  
  name [ James Bond ],  
  address [ London ],  
  bidderID [ 007 ]  
} t=[2,2]
```

```
itemDescription {  
  auctionID [ 123 ],  
  itemID [ 789 ],  
  name [ sofa ],  
  value [ 1000 ]  
} t=[3,4]
```

```
bid {  
  auctionID [ 123 ],  
  itemID [ 789 ],  
  bidderID [ 007 ],  
  value [ 1001 ]  
} t=[5,5]
```

```
1HammerBeat {  
  auctionID [ 123 ],  
  itemID [ 789 ],  
  bidderID [ 007 ],  
  value [ 1001 ]  
} t=[6,6]
```

```
2HammerBeat {  
  ...  
} t=[7,7]
```

```
3HammerBeat {  
  ...  
} t=[8,8]
```

```
event s: sell {  
  auctionID [ 123 ],  
  itemID [ 789 ],  
  bidderID [ 007 ],  
  value [ 1001 ]  
} t=[9,10]
```

**while s:** ...

$b(s) = 9$

$e(s) = 10$

# Event Stream Model

Idea: Event Stream Model captures application-specific knowledge and provides queries with it

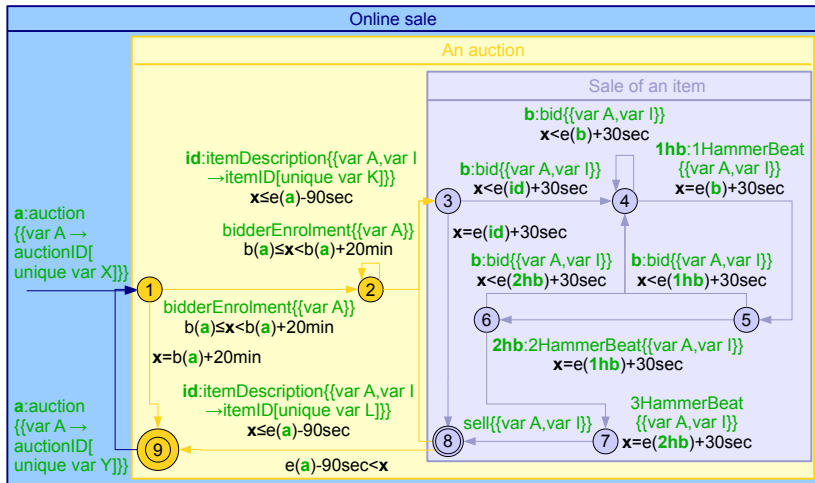
Event Stream Model:

1. Timed Finite Automaton
2. Constraints
3. Queries

Impact:

- Simple queries
- Recognition of unsatisfiable (sub-) queries
  - Deletion/suspension of these (sub-) queries
  - Non-storage of irrelevant events
- Sharing of results of *equivalent* subqueries of many queries

# Timed Finite Automaton (R.Alur, D.Dill, 1990)



$x$  – global clock

$b(a)$  – beginning of the recent event with identifier  $a$

$e(a)$  – end of the recent event with identifier  $a$

# Constraints

Price rises during an auction

```
1 FOR ALL itemDescription {{ auctionID [ var A ], itemID [ var I ], value [ var S ] }}
2   FOR ALL bid {{ auctionID [ var A ], itemID [ var I ], value [ var V ] }}
3     FOR ALL sell {{ auctionID [ var A ], itemID [ var I ], value [ var E ] }}
4       var E  $\geq$  var V, var V  $\geq$  var S
5     END
6   END
7 END
8
9 FOR ALL event  $b_1$ : bid {{ auctionID [ var A ], itemID [ var I ], value [ var  $V_1$  ] }}
10  FOR ALL event  $b_2$ : bid {{ auctionID [ var A ], itemID [ var I ], value [ var  $V_2$  ] }}
11    IF  $b_2$  after  $b_1$ 
12      THEN var  $V_2 >$  var  $V_1$ 
13    END
14  END
15 END
```

# Event Stream Model

Idea: Event Stream Model captures application-specific knowledge and provides queries with it

Naive ways:

## 1. Query compilation

- completion of all queries with application-specific knowledge before the evaluation
- happens only once
- Event Stream Model is not necessary during the evaluation
- A compiled query is evaluated **always**

## 2. Query interpretation

- providence of each query with application-specific knowledge during its evaluation
- happens everytime a query is called
- Event Stream Model is analysed during the evaluation of each query
- An interpreted query is evaluated only **if it is satisfiable**

## Naive Query Evaluation

	Without Semantic Optimisation	With Semantic Optimisation	
		Naive Query Compilation	Naive Query Interpretation
Costs for semantic query optimisation	no	ones for each query	ones for each query call
Query evaluation time	always	always	if it is satisfiable
Number of relevant events	few	many	few
Storage of irrelevant events	yes	no	no

# Context-dependent Query Compilation

## Ideas:

- Completion of all queries with application-specific knowledge before the evaluation ⇒
    - Happens only once for each query
    - No irrelevant events are saved
  - **the whole** Event Stream Model is analysed during the evaluation to save **the current state of each process** ⇒
    - Each query is evaluated only **if it is satisfiable**
    - **A smaller part** of the Event Stream Model is relevant for the compilation of each query ⇒
      - **simpler** query compilation
      - **smaller** compiled queries
      - **fewer relevant events** must be saved
- compared to naive query compilation

# Summary

Goal: Semantic Query Optimisation

Means: Event Stream Model:

1. Timed Finite Automaton
2. Constraints
3. Queries

+ Context-dependent Query Compilation

Impact:

- Simple queries
- Recognition of unsatisfiable (sub-) queries
  - Deletion/suspension of these (sub-) queries
  - Non-storage of irrelevant events
- Sharing of results of *equivalent* subqueries of many queries

# Work Plan

1. Survey of Event Query Languages
2. Lessons learnt from Use Cases
3. Event Stream Model
  - Related work
  - Constraint language
  - Complexity
4. Query planner for CEP differs from a query planner for databases because it
  - works with other *operators*
  - *incrementally* evaluates the queries
  - cares about *garbage collection* of events
  - considers *causality* of events
  - allows *expensive* query optimisation

# Work Plan

1. Survey of Event Query Languages
2. Lessons learnt from Use Cases
3. Event Stream Model
  - Related work
  - Constraint language
  - Complexity
4. Query planner should:
  - link the Event Stream Model with XChange<sup>EQ</sup>,
  - suggest many query plans and
  - choose the best one
5. Experimental analysis

Main goal: Combined usage of different semantic query optimisation techniques in CEP