BP-Mon:

Monitoring Business Processes with Queries

Catriel Beeri, Anat Eyal, Tova Milo, Alon Pilberg

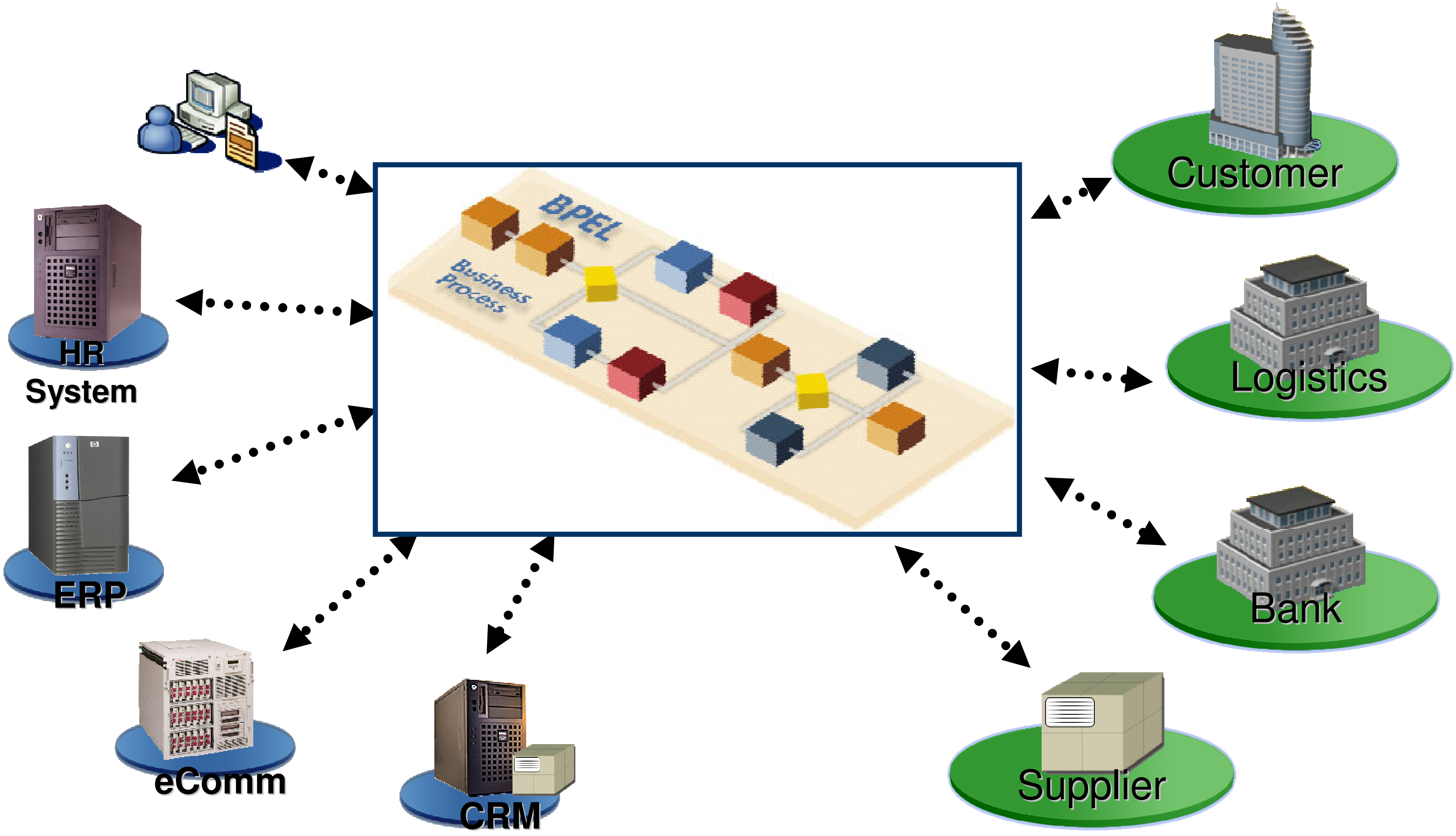
Hebrew University

Tel-Aviv University

Outline

- Introduction to Business Processes
- Overview of BP-Mon by example
- Formal model
- Implementation & experiments
- Summary

Business Processes



BPEL in a Nutshell

Business Processes Execution Language (BPEL)

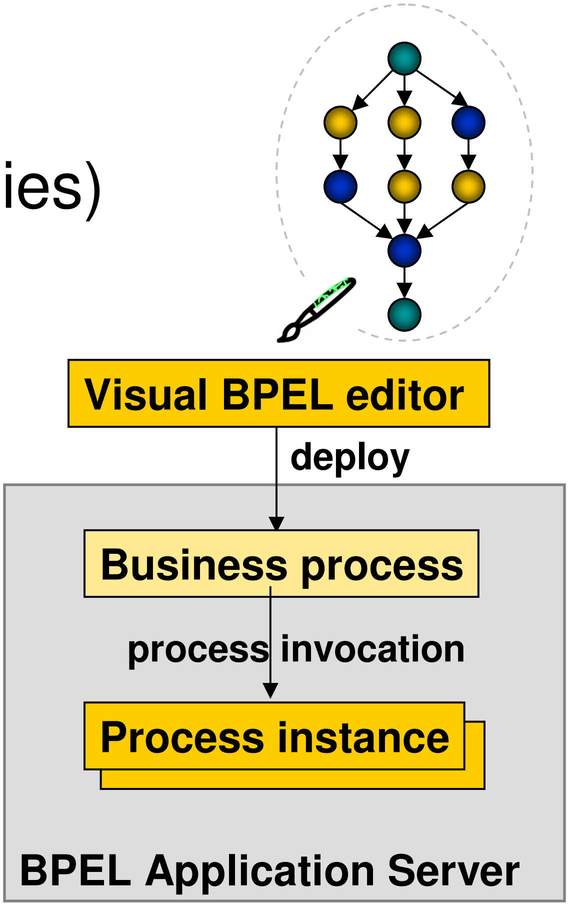
Process spec. represented in XML

- operations (atomic/ compound activities)
- flow and data

Designed using visual tools

- graphs of nodes and edges

Compiled into executable code & run on **<u>any</u>** BPEL application server



Visual BPEL editor

deploy

Business process

process invocation

Process instance

BPEL Application Server

high-level & portable

The Need for Monitoring

Imagine you run an auction service…

- **Guarantee fair play:** notify on too many cancels
- **Maintain SLA:** monitor response time
- **Promotions:** prizes for the x10,000 transaction
- **Illegal access**: notify on buyers attempt to confirm bids without registering first

Monitoring is crucial for enforcing business policies and meeting efficiency & reliability goals

Background and Challenges

BPM systems send process traces as events

Very large field: active database, publish-subscribe, composite events, temporal logic,…

	Shortcoming of current approaches	BPEL challenges
Abstraction level	•Two levels: events vs. spec	→ Write queries the same way as the spec
Efficiency	•Generic optimizations	→ Exploit knowledge of the spec
Implementation & Deployment	•Propriety language •Not portable	→ Declarative language Run everywhere

Contributions

Abstraction level

- High level graphical query language
- Tight analogy to the spec

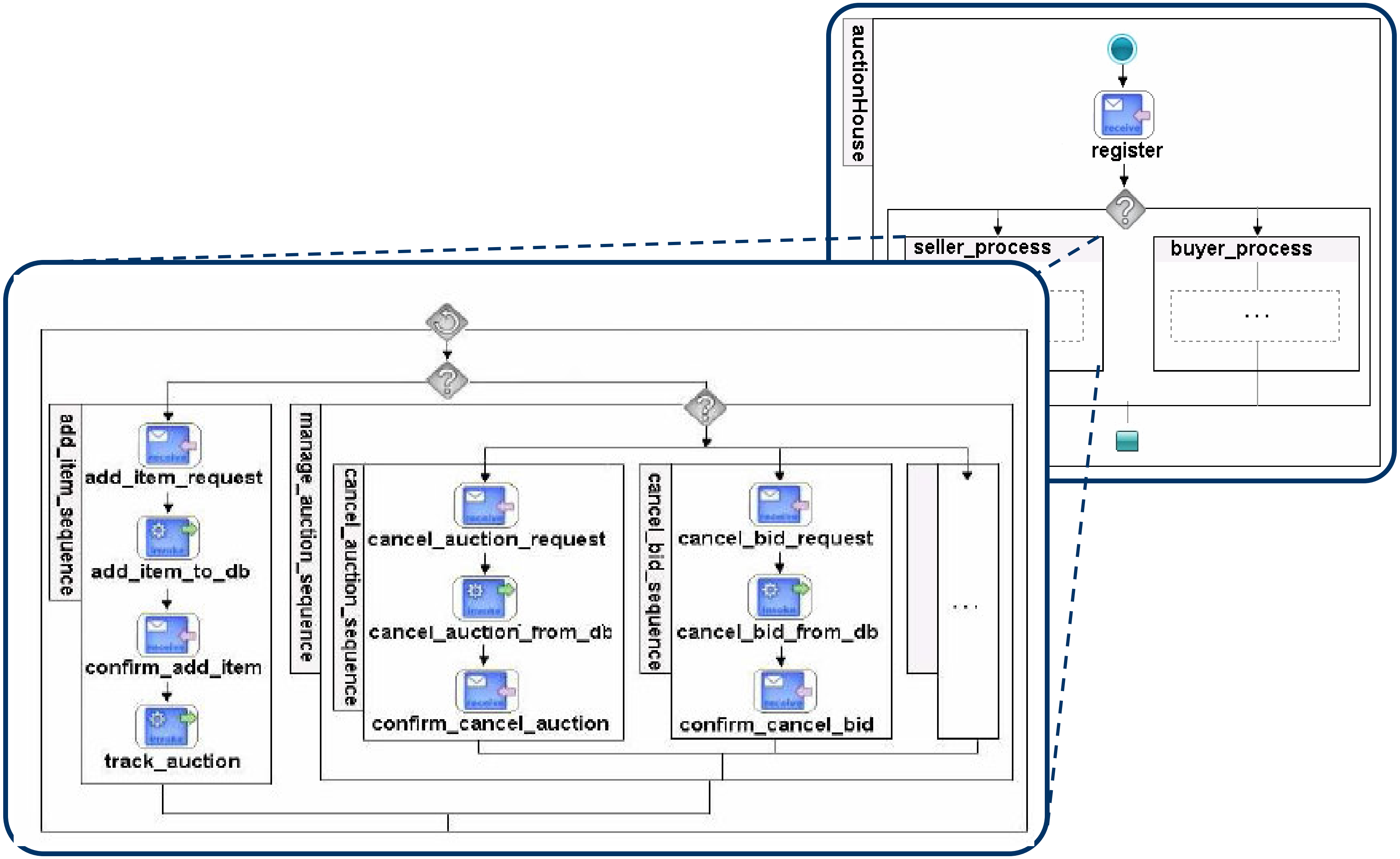
Efficiency

- Dedicated efficient automata based Algorithm
- Novel optimizations based on analysis of spec
- Pruning of redundant monitoring

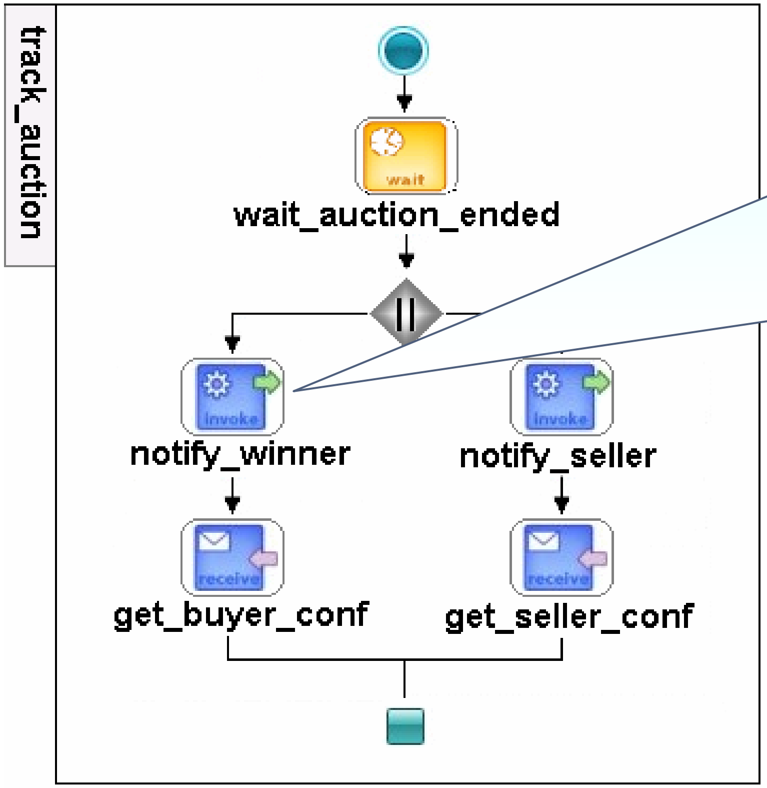
Implementation & deployment

- Compiles a BP-Mon query into a BPEL process
- Easy deployment, portability
- Minimal overhead

Running Example



Events

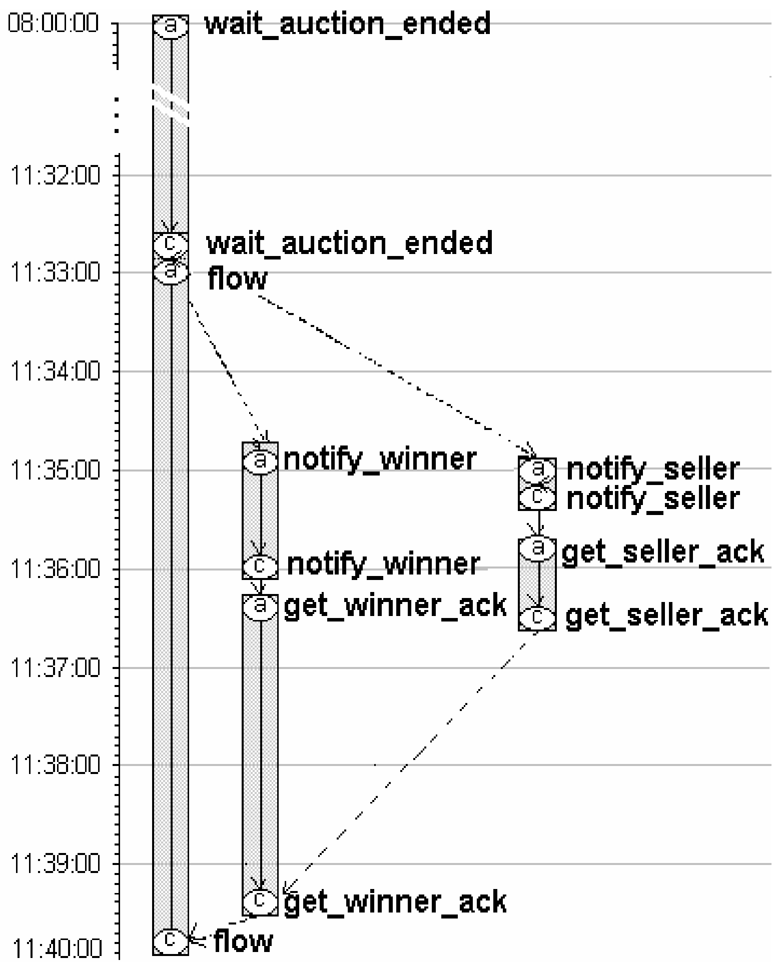


```
<actionData>
  <header>
    <processName> auctionHouse </processName>
    <instanceId> 517 </instanceId>
    <sensorTarget> notify_winner </sensorTarget>
    <timestamp> 2006-05-31T11:32:46.510+00:00 </>
  </header>
         …
  <activityData>
     <activityType>invoke </activityType>
     <evalPoint> completion </evalPoint> …
```

BP Execution Traces as DAGs

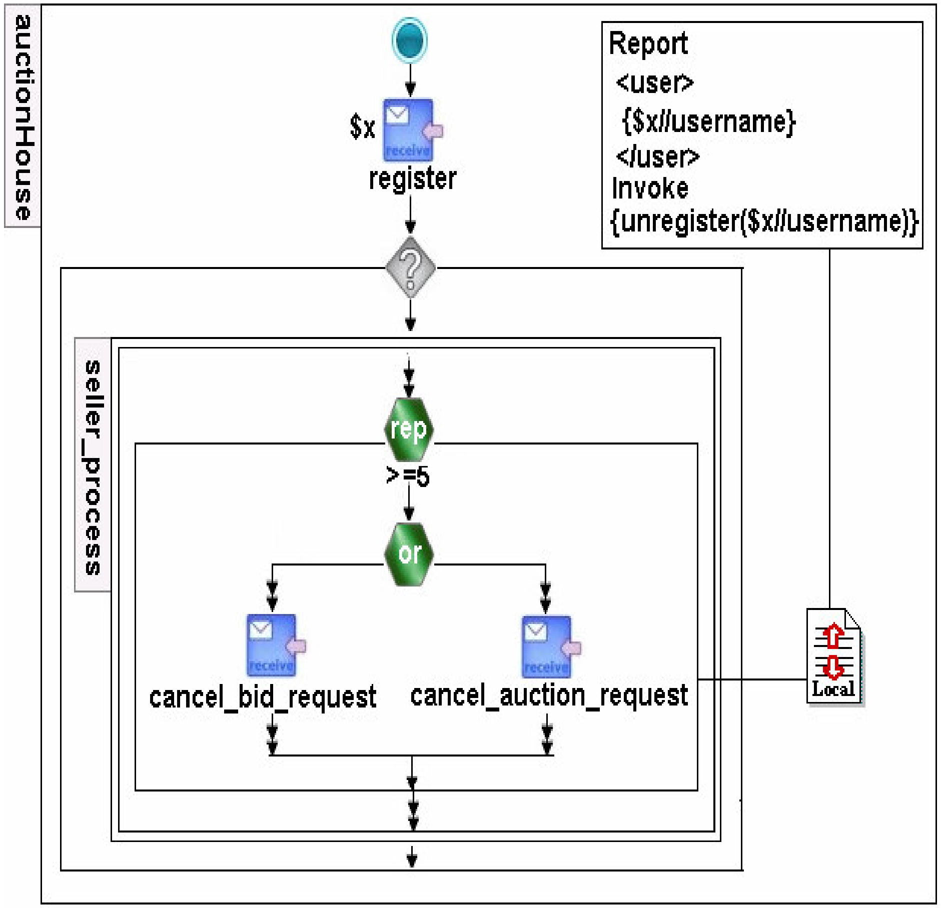
Nested set of DAGs:

- Nodes
  - Activation ⓐ
  - Completion ⓒ
- Timestamps
- Edges
  - Flow →
  - Zoom-in, Zoom-out ⇢



Query Example (1)

Too many Cancels



Use execution patterns

- Transitive edges

  ↓ ↓

- Transitive nodes

  □ ▣

- Regular expressions

  or rep

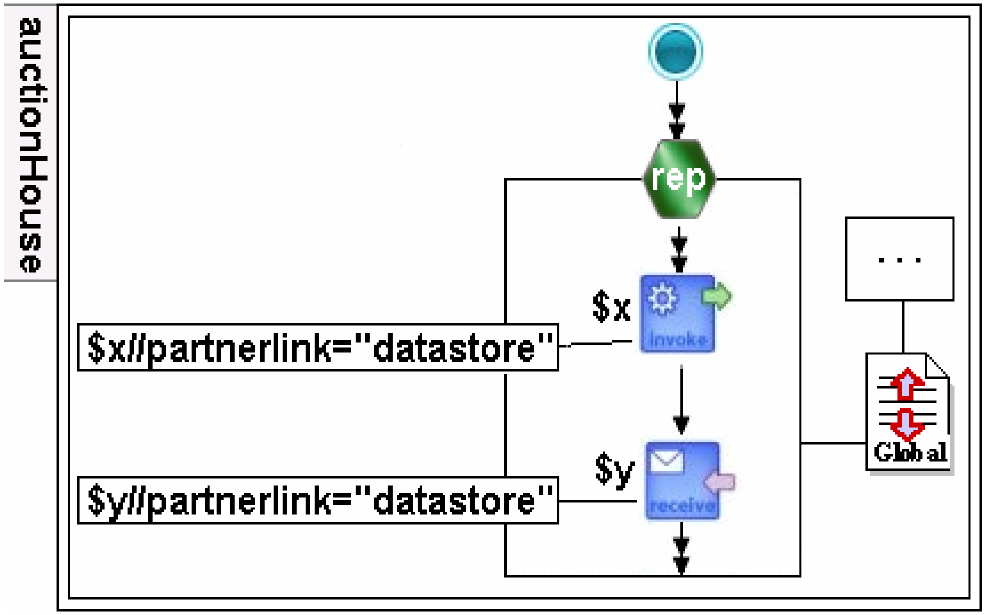
- Report/ Report*

BP-Mon by Example

Query Example (2)

Monitor response time

(and notify the process to change the db)



Sliding window

- Time based

  Report* Every 1 hrs Range 2 hrs

- Instance based

  Every 100 entries Range 200 …

Report

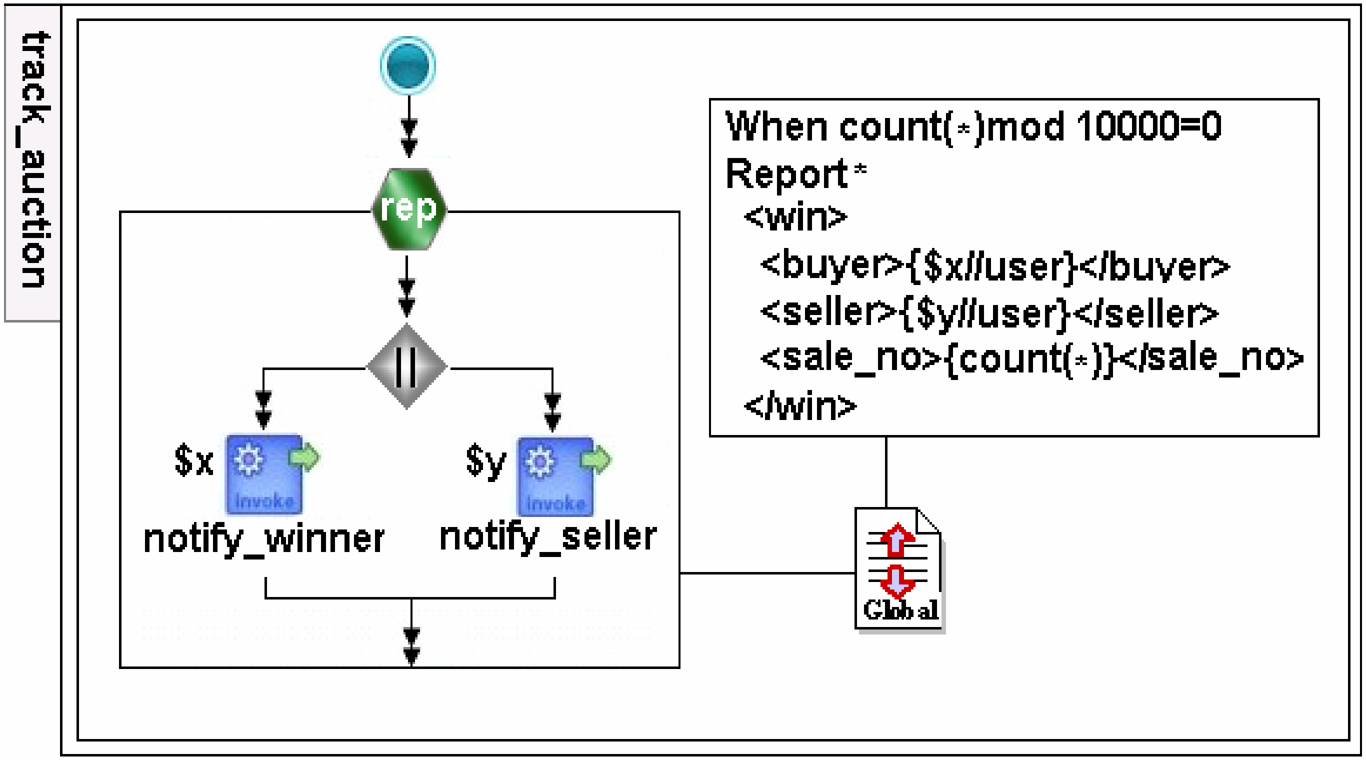
- Local/Global
- Multiple reports

Output format

- XQuery like
- Group by having

Query Example (3)

X10,000 win

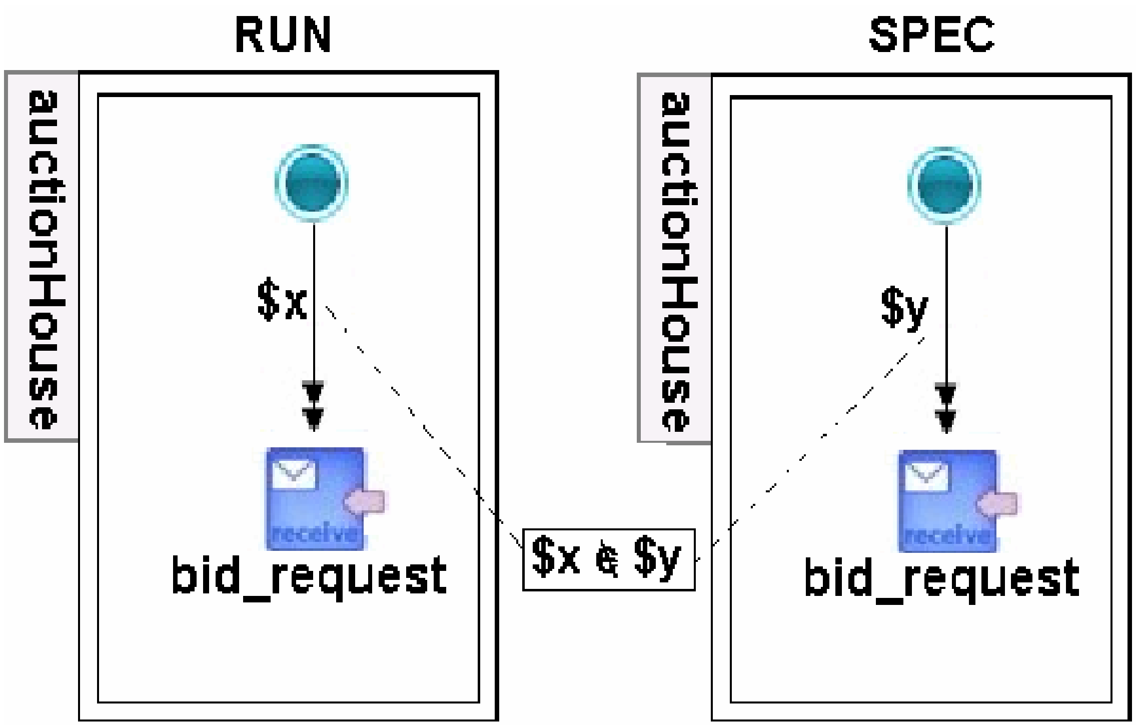


```
When count(*)mod 10000=0
Report*
 <win>
  <buyer>{$x//user}</buver>
  <seller>{$y//user}</seller>
  <sale_no>{count(*)}</sale_no>
 </win>
```

BP-Mon by Example

Query Example (4)

Static and dynamic analysis



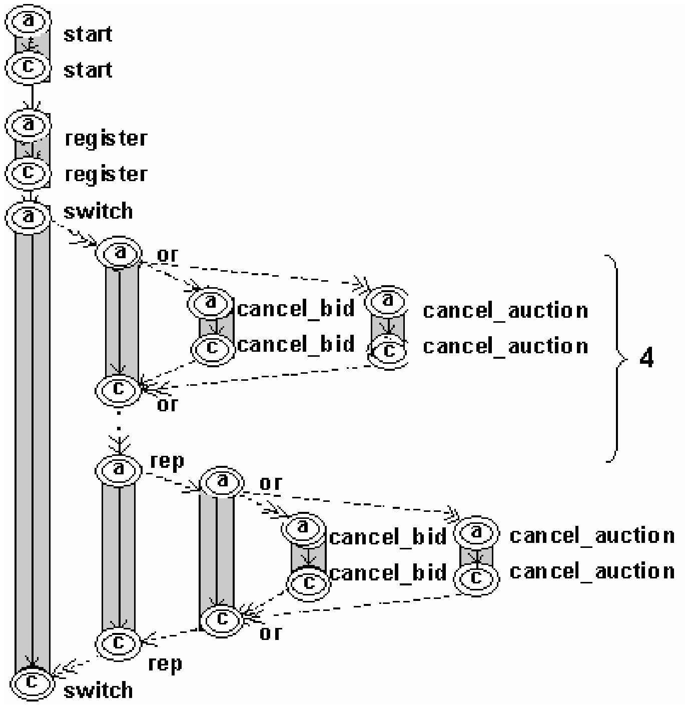
Formal Model

Queries: Execution Patterns

- EX-trace: nested DAGs
- EX-pattern: EX- trace without timestamps

  transitive edges & nodes

  ‘any’, ‘or’, ‘rep’



A query defines a set of concrete Ex-patterns obtained by:

- Rep– replacing with arbitrary number of copies
- Or – choosing an internal trace & replacing

Formal Model

An Embedding

p concrete EX-pattern, *e* EX-trace.

Definition: An embedding of *p* into *e* is a homomorphism from the nodes/edges in *p* to nodes/edges/paths in *e* s.t.

- node labels match
- edge (transitive)→edge(path), of the same type
- direct edge of transitive nodes → any type

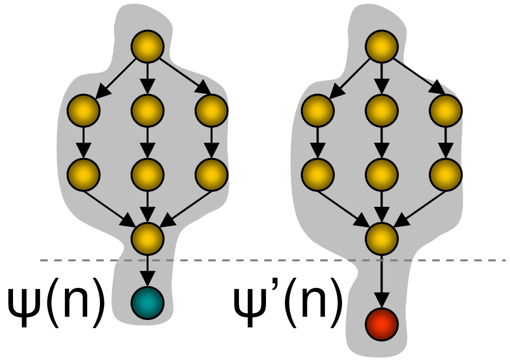
Formal Model

A Greedy Embedding

p EX-pattern, *e* EX-trace, *S* a set of embeddings of patterns in *concrete(p)* into *e*

Definition(semi-formal): $\psi \in S$ is greedy (in *S*) if:

There is no other embedding ψ' that agrees with ψ on the prefix of $n \in p$ but matches n with an earlier timestamp



The Algorithm

- Incrementally extends a greedy embedding to one of a larger prefix
- Automaton with Ex-pattern nodes as states
  - Tries to match (concurrently) the concrete patterns of the given EX-pattern
  - Attempts to match events as early as possible
  - On failure: backtracks & retries

Complexity: polynomial in the size of the trace
(with the exponent determined by the size of the pattern)

The Algorithm (cont)

Non-deterministic automaton

- Manage simultaneously a large number of active states

Deterministic automaton

- Potential exponential growth in the size of the automaton

⇒ We provide a hybrid solution

- Lazy DFA
- Small automaton, same size as the pattern
- Relatively few states are simultaneously active

Issues:

- Backtracking
- Retaining of events

Optimization: Irrelevance & Inconsistency

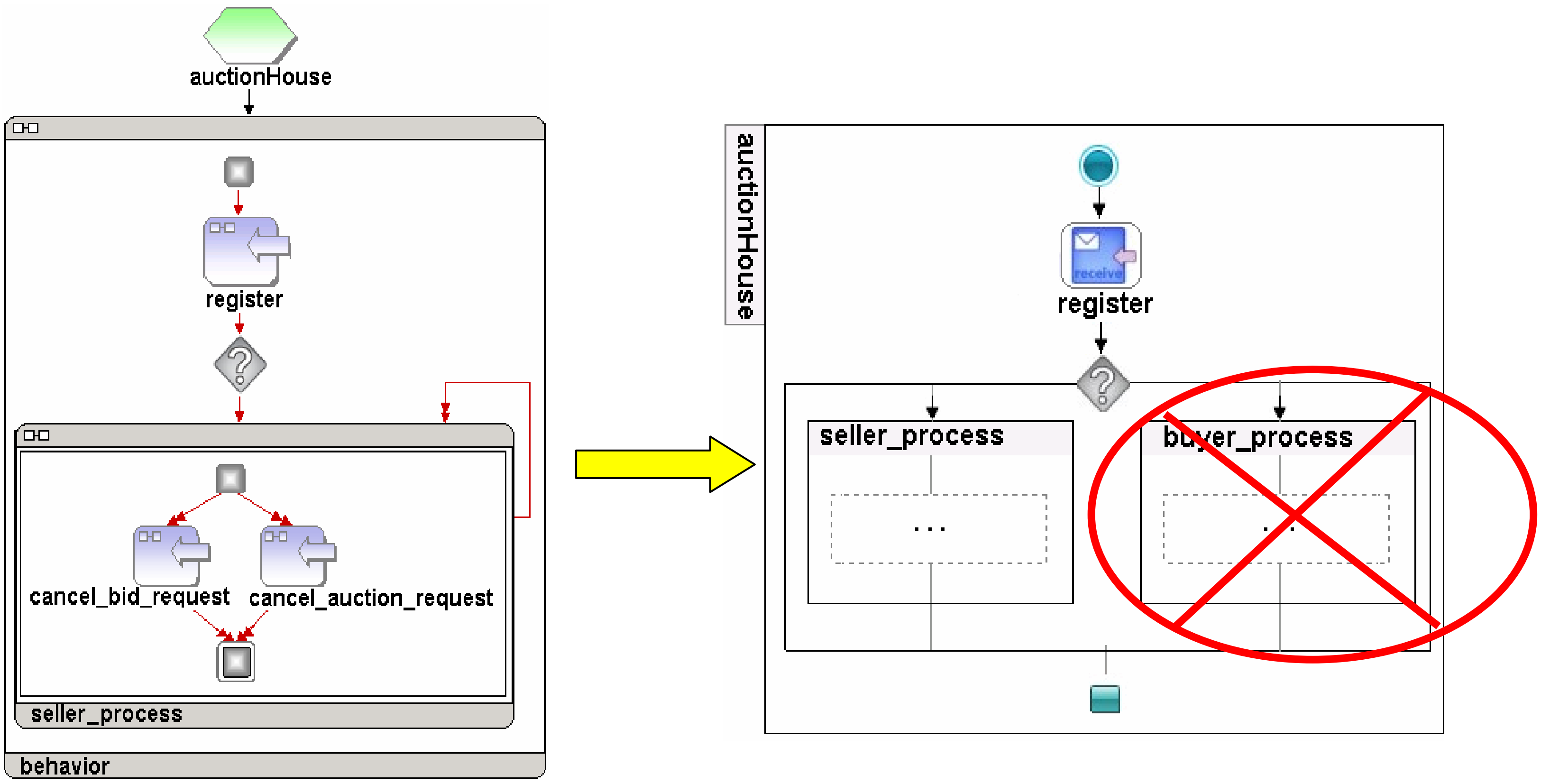
Let *S* be a BP specification, *o* an activity in *S*

Definitions (semi-formal):

- Activity *o* is irrelevant to query node *n*
  - if there is no EX-trace of *S* where it participates in an embedding.
- Activity *o* is inconsistent with EX-pattern *p*
  - if *p* cannot be embedded into any EX-trace of *S* that contains an activation event of *o*.

Algo uses BP-QL [VLDB06] for spec analysis

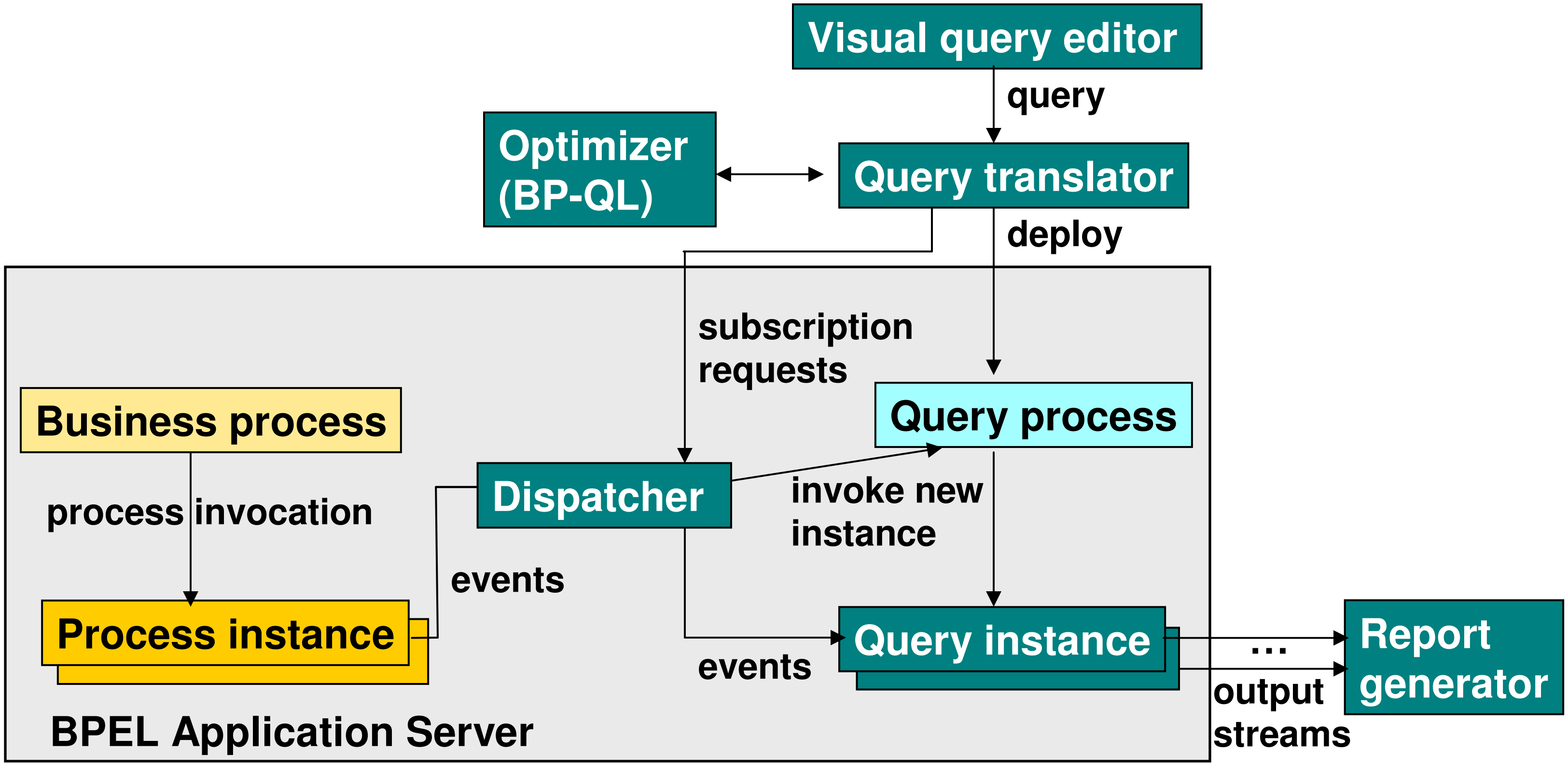
Optimization (example)



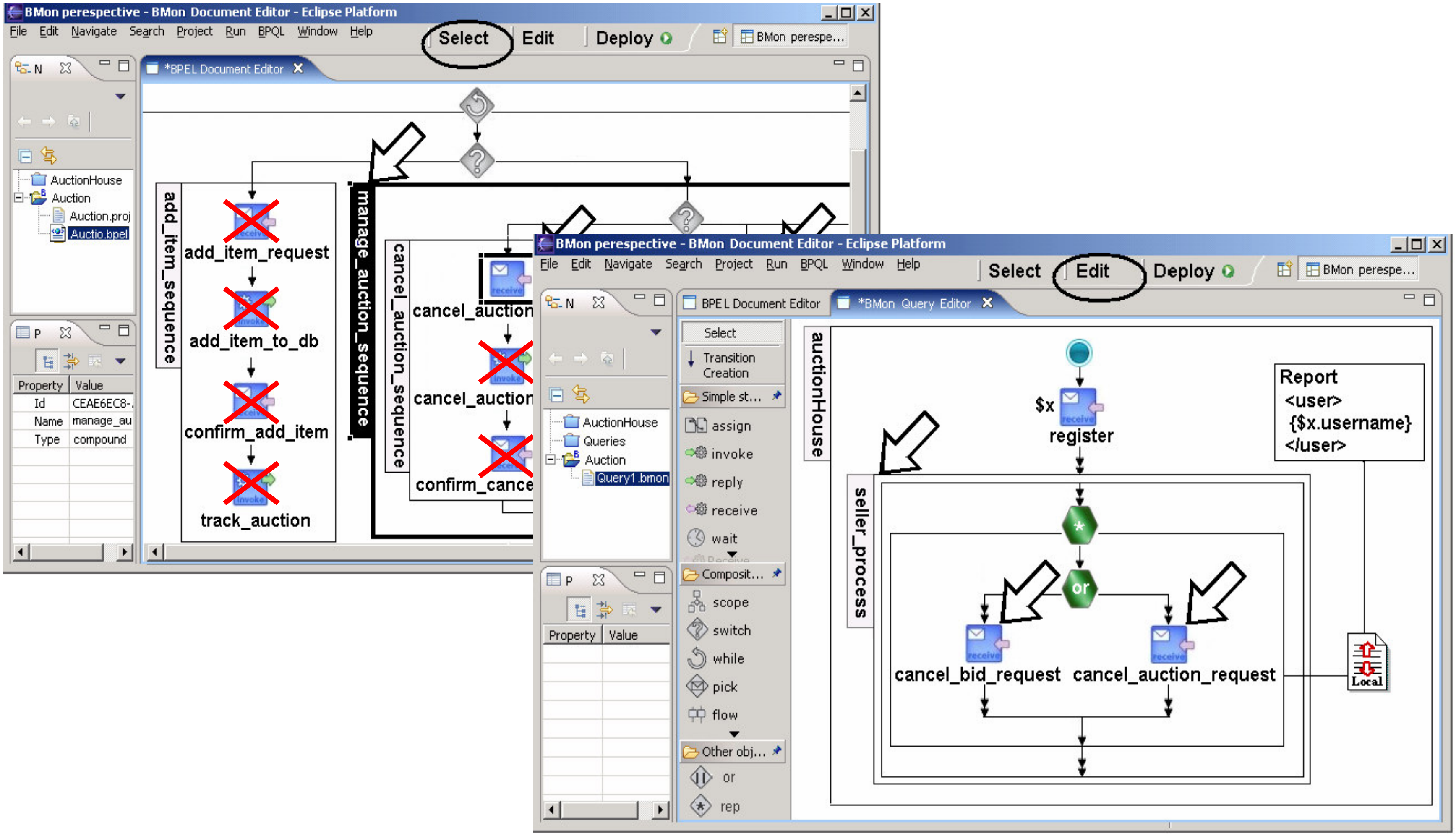
BP-QL query (on spec)

Architecture

A monitoring query is compiled into a BPEL process



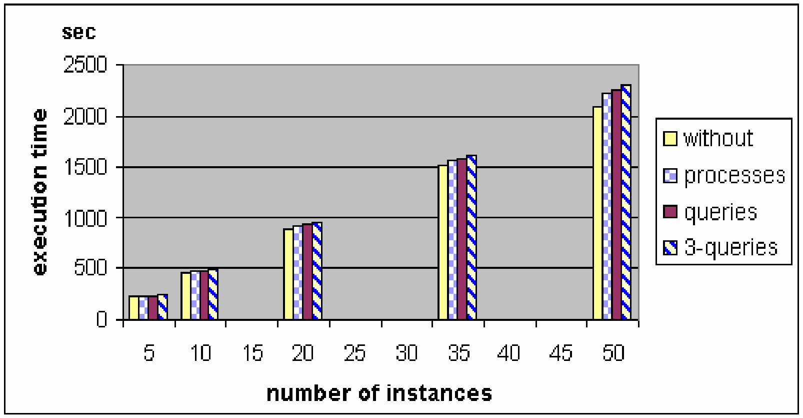
Visual Interface



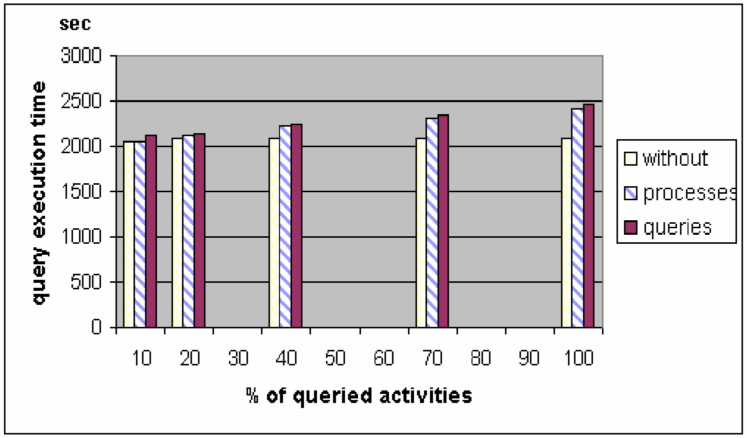
Property	Value
Id	CEAE6EC8-.
Name	manage_au
Type	compound

Implementation

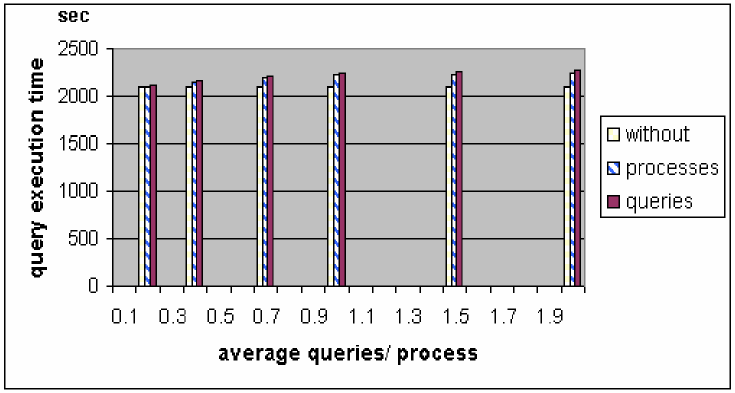
Experiments Results



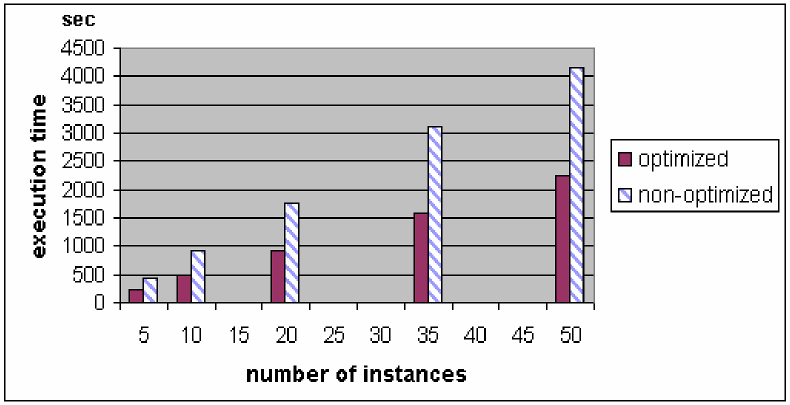
Queries overhead



Effect of % of queried activities



Effect of queries per process



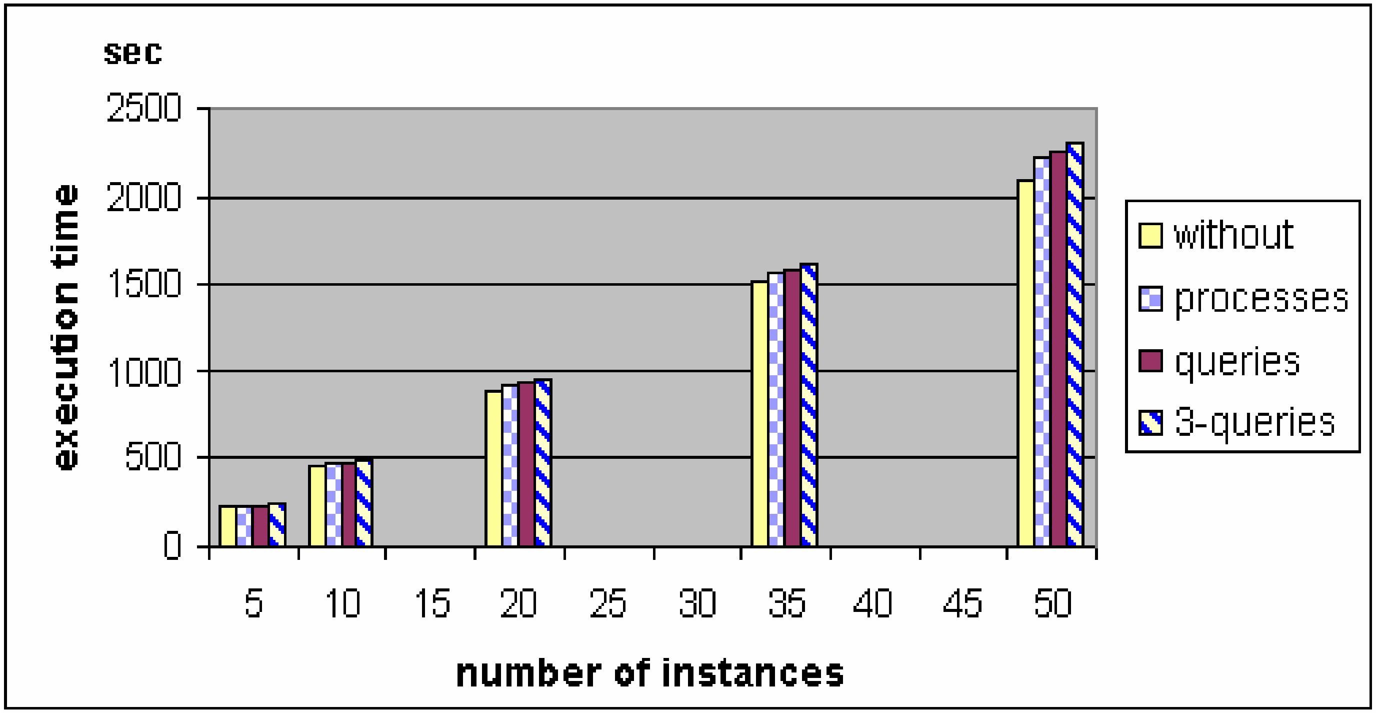
Impact of optimization

Implementation

Experiments Results: Queries Overhead

Each process: 200 activities, 40% queried

Query: report*, 3 reports



Conclusion

- User friendly query language for monitoring BPs:
  - Graphical and intuitive (wizard)
- Semantics:
  - Early match (greedy), all matches
- Algorithm
  - Lazy DFA
  - Irrelevancy & inconsistency
- Implementation
  - Compiles into BPEL=>

    Easy deployment, portability, and minimal overhead

Ongoing and Future Work

- Querying/mining logs
- Incomplete information
- Application to software monitoring and verification
- More optimization

Thank you !