Publish/Subscribe

Hans-Arno Jacobsen

Bell University Laboratory Chair in Software Engineering
Middleware Systems Research Group
University of Toronto
Amazon to Chapters to you ... .

Monday, October 10th  in Cyberspace

Thursday, November 15th, in Toronto

Your book “...” is available at .... $10 off
Business Process Execution & Web Service Composition

- Business Process:
  - Receive
  - Assign
  - Flow
  - Pick
  - Invoke
  - Scope
  - Wait
  - Reply

- Business Process:
  - Receive
  - Switch
  - Scope
  - Reply

Components:
- Client
- WS
- Agent
- Broker
- Database
Other “Similar” Applications

- Selective information dissemination
- Location-based services
- Personalization
- Alerting services
- Application integration
- Job scheduling
- Monitoring, surveillance, and control
- Network and distributed system management
- Workforce management
- (Scientific) workload management
- Business activity monitoring
- Business process management, monitoring, and execution
What Relates All These Applications?

- Asynchronous state transitions captured as events that
  - drive and underlay
    - all applications and
    - infrastructures implementing these applications
- Require middleware support for event processing
- Publish/Subscribe is ideally suited to fulfill these requirements
Publish/Subscribe
Publish/Subscribe

Stock markets

Publisher

Publisher

Broker(s)

Notifications

Subscriptions

Notifications

Subscriptions

Publisher

Subscriber

Subscriber

Subscriptions:
IBM > 85
ORCL < 10
JNJ > 60

AMGN=58
ORCL=12
HON=24
MSFT=27
IBM=84
INTC=19
JNJ=58

TSX

NYSE

NASDAQ

Stock markets

Publish/Subscribe Lecture
That’s Like Data Base Querying 😞 !!

Query and subscription is very similar. Set of tuples and publication is very similar.

However, the two problem statements are inverse.
Key Benefit of Publish/Subscribe

- Decoupling of publishers and subscribers
  - Publishers do not need to know subscribers
  - Publishers and subscribers do not need to be up simultaneously
  - Amenable for physical distribution
Benefits of Publish/Subscribe

- **independence** of participants
- lends itself well to distributed system development
  - de-coupled development & processing
  - (dynamic) system evolution
- **interaction** with **large number** of entities facilitated
- naturally supports **non-continuous** operations
- potential for scalability & fault-tolerance
- open for (legacy) **system integration** on either end

Of course it is not a **one size fits all** paradigm, but a good solution for certain kinds of problems.
Publish/Subscribe Matching Problem

- Given a set of subscriptions, $S$, and a publication, $e$, return all $s$ in $S$ matched by $e$.
- $e$ is referred to as **event** or **publication**
- Splitting hairs
  - *Event* is a state transition of interest in the environment
  - *Publication* is the information about $e$ submitted to the publish/subscribe system
- Simple problem statement, widely applicable, and lots of open questions
Problem Instantiations I

- Text / search strings (information filtering)
- Semi-structured data / queries
  - attribute-value pairs / attribute-operator-value-predicates
  - XML, HTML
- Tree-structured data / path expressions
  - XML ./ XPath expressions
- Graph-structured data / graph queries
  - RDF / RDF queries (e.g., SPARQL)
- Regular languages / regular expressions
- Tables / SQL queries
Problem Instantiations II

- Different matching semantics
  - Crisp
  - Approximate,
  - Similar
  - n-of-m (n of m predicates match)
  - Probability of match
Problem Instantiations III

- Centralized and distributed instantiation
- Networking architecture
  - Internet (as overlay network)
  - Peer-to-peer style interface (DHT, table-lookup)
  - With mobile publishers, subscribers, brokers
  - Ad hoc network
Publish/Subscribe Models

- Channel-based model
  - Subscribe & publish to a channel
- Topic-based model
  - … topics and topic hierarchy
- Type-based model
  - … typed objects
- Content-based model
  - … to content of messages
- Subject-spaces (State-based model)
  - Maintain state in publications and subscriptions
Channel-based

Publisher

Publisher

Broadcast Channel

Subscriber

Subscriber

Subscriber
Topic-based publish/subscribe
The Content-based Model

Language and Data model

- Conjunctive Boolean functions over predicates
- Predicates are attribute-operator-value triples
  - `[class,=,trigger]`
- Subscriptions are conjunctions of predicates
  - `[class,=,trigger],[appl,=,payroll],[gid,=,g001]`
- Publications are sets of attribute-value pairs
  - `[class,trigger],[appl,printer],[gid,g007]`

Matching semantic

- A subscription matches if all its predicates are matched
Content-based routing

- Distributed publish/subscribe
- **Network of** publish/subscribe brokers
- Subscriptions & publications are injected into network at closest edge broker
- Routing protocol distributes subscriptions throughout network
- Network **routes relevant publications to interested subscribers**
- Routing is based on content; it is not based on addresses, which are not available
- Subscriptions may change dynamically
Content-based Routing

1. Advertise
2. Subscribe
3. Publish

A: [class, =, stock], [name, =, HP], [price, >, 50]
S: [class, =, stock], [name, =, *], [price, >, 50]
P: [class, stock], [name, *], [price, 50]

- Event-Based
- Content Routing
- Flexible
- Decoupled
- Declarative
- Responsive
Applications

Supply chain and logistics

Service oriented architecture

RFID and sensor networks

Workflows, business processes and job scheduling

Event-Based

- In flight
- Delivered
- Order
- Job A done
- Trigger
- Fault
- Callback
- Invoke Loan
- Transform
- Razor SKU
- Light
- Temperature
Publish/Subscribe in Industry

- Standards
  - CORBA Event Service
  - CORBA Notification Service
  - OMG Data Dissemination Service
  - Java Messaging Service
  - WS Eventing
  - WS Notification
  - INFO-D (Grid Forum)

- Emerging technologies
  - RSS aggregators
    - PubSub.com, FeedTree
  - Real-time data dissemination
    - TIBCO, RTI Inc., Mantara Software
  - Application integration
    - Softwired
  - Hardware-based brokers
    - Sarvega (Intel), Solace Systems, DataPower (IBM)
Publish/Subscribe in Academia

- Research projects
  - Elvin (Australia)
  - Gryphon (IBM)
  - Hermes (Cambridge)
  - SIENA (Boulder)
  - REBECA (Darmstadt)
  - ToPSS (UofT)
  - PADRES (UofT)
The Toronto Publish/Subscribe System Family (ToPSS)

- Matching algorithms
  - Language expressiveness, scalability, speed

- Routing protocols
  - Network architectures, scalability

- Higher level abstractions
  - Workflow execution
  - Monitoring

<table>
<thead>
<tr>
<th>ToPSS</th>
<th>A-ToPSS</th>
<th>CS-ToPSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(matching)</td>
<td>(approximate)</td>
<td>(composite subs)</td>
</tr>
<tr>
<td>S-ToPSS</td>
<td>L-ToPSS</td>
<td>Rb-ToPSS</td>
</tr>
<tr>
<td>(semantic)</td>
<td>(location-based)</td>
<td>(rule-based)</td>
</tr>
<tr>
<td>X-ToPSS</td>
<td>persistent-ToPSS</td>
<td></td>
</tr>
<tr>
<td>(XML matching)</td>
<td>(subject spaces)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>M-ToPSS</th>
<th>P2P-ToPSS</th>
<th>LB-ToPSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(mobile)</td>
<td>(peer-to-peer)</td>
<td>(load balancing)</td>
</tr>
</tbody>
</table>

Federated-ToPSS
(federation of ToPSS brokers)
Ad hoc-ToPSS
(ad hoc networking)

| Historic-ToPSS | FT-ToPSS |
| (historic data) | (fault tolerance) |
| JS-ToPSS | BPEL-ToPSS |
| (job scheduling) | (BPEL execution) |
Overall Project Vision

A Real-Time Event-driven Enterprise
Matching and Content-based Routing

Routing Tables

input queue

output queues

P → S

A

S

P

S

P

S

Publish/Subscribe Lecture
Matching Algorithms

- For solving the pub/sub matching problem

- Tree-based algorithms
- Graph-based algorithms
- Automaton-based algorithms (NFA, DFA)
- Two-staged algorithms
  - predicate matching
  - subscription matching
Predicate Matching
Predicate matching problem

- Given a set \( P \) of predicates and an event \( e \), identify all predicates \( p \) of \( P \) which evaluate to true under \( e \).

Example:

\[
e = \{..., (\text{price, 5}), (\text{color, white}) \ldots\}
\]

\[
p1: \text{price} > 5; \quad p2: \text{color} = \text{red}; \quad p3: \text{price} < 4
\]

\[
p1 \text{ is false} \quad p2 \text{ is false} \quad p3 \text{ is true}
\]

Predicate bit vector:

\[
\begin{array}{cccc}
0 & 0 & 1 & \ldots \\
p1 & p2 & p3 & \text{predicate IDs}
\end{array}
\]
Data structure overview

- hash table on attribute name

\[ e = \{\ldots, (\text{price}, 5), \ldots\} \]

<table>
<thead>
<tr>
<th></th>
<th>price</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>\ldots</td>
<td>\ldots</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

predicate index
Predicate Index

- one ordered linked list per operator
- insert, delete, match are $O(n)$-operations (per attribute name in $e$ and per operator)
- alternatively, use a B-tree or B+-tree etc.
Observations

- countable domain types with small cardinality
  - integer intervals
  - collections (enums)
  - a set of tags

- Examples
  - price : \([0, 1000]\), models variety of prices
  - color, city, state, country, size, weight
  - all tags defined in a given DTD
  - predicate domain is often context dependant, but limited in size
    - prices of cars vs. prices of groceries
Predicate Matching for Finite Domains

Price : [0, 1000]  
\[ e = \{\ldots, (\text{price}, 5), \ldots\} \]

\[
\begin{array}{cccccc}
& 0 & 1 & 2 & 3 & \ldots & 6 & \ldots & 1000 \\
= & p17 & & & & p4 & & & \\
< & p11 & & & & p3 & & & \\
> & & p6 & & & p1 & & & \\
! &= & p7 & & & p9 & & & \\
\end{array}
\]

\( p1: \text{price} > 5; \quad p3: \text{price} < 4; \quad p4: \text{price} = 5; \quad p9: \text{price} \neq 5 \ldots \)
Predicate Matching Symmetries

price : [0, 1000]

e = {..., (price, 5), ...}

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>6</th>
<th>...</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>&gt;</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>&lt;</td>
<td></td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

Publish/Subscribe Lecture 34
Experiments and evaluation
Predicate Matching Performance (list-based scheme)

domain sizes: 250, 10,000, 100,000

Matching Time (ms)

Number of Distinct Predicates

500 K 4.5 M
Predicate Matching Performance
(table-based scheme)

domain sizes: 250, 10,000, 100,000

Matching Time (ms)

Number of Distinct Predicates

500K

4.5M
Predicate Matching Performance
(tables-based vs. list-based scheme)

for the mixed domain

![Graph showing matching time vs. number of distinct predicates]

Matching Time (ms)

0 200 400 600 800 1000 1200

500000 1e+06 1.5e+06 2e+06 2.5e+06 3e+06 3.5e+06 4e+06 4.5e+06

Number of Distinct Predicates

4.5 M
Predicate Matching Data Structure Size (table-based vs. list-based scheme)

1.4 GB

20 MB

Memory Use (bytes)

Number of Distinct Predicates

Table-based evaluation
Link-list-based evaluation

4 M
Subscription Matching
Multiple one-dimension indexes

- One-dimension indexes.
  - hash tables
  - B-trees
  - Interval Skip Lists
- Counting algorithm
- Hanson algorithm
- Propagation algorithm
Counting algorithm

- Subscriptions consist of a set of predicates
  - S1: $(2< A<4) \land (B=6) \land (C >4) \Rightarrow pA : (2< A<4), pB (B=6), pC:(C>4)$
  - S1: $(2< A<4) \land (C=3) \Rightarrow pA : (2< A<4) pC: (C=3)$

- A Subscription matches the event if all its predicates are satisfied.

- **Idea:** Count the number of satisfied predicates per subscription
Data structures for the counting algorithm

Indexes

A

B

C

Indexes

S1: p1, p2, p4
S2: p1, p3
S3: p3, p5

TOTAL NUMBER
S1 3
S2 2
S3 2

COUNT
S1 0
S2 0
S3 0

Predicate vector

p1
S1 S2
p2
S1
p3
S2 S3
p4
S1
p5
S3

Publish/Subscribe Lecture
Counting algorithm

Indexes

<table>
<thead>
<tr>
<th>Index A</th>
<th>Index B</th>
<th>Index C</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>p2</td>
<td>p3</td>
</tr>
<tr>
<td>S1</td>
<td>S1</td>
<td>S1</td>
</tr>
<tr>
<td>S2</td>
<td>S2</td>
<td>S3</td>
</tr>
<tr>
<td>S3</td>
<td>p3</td>
<td>p4</td>
</tr>
<tr>
<td></td>
<td>p5</td>
<td></td>
</tr>
</tbody>
</table>

Predicate vector

E: (A,5),(B,6)

TOTAL NUMBER

<table>
<thead>
<tr>
<th>S1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>2</td>
</tr>
<tr>
<td>S3</td>
<td>2</td>
</tr>
</tbody>
</table>

COUNT

<table>
<thead>
<tr>
<th>S1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>2</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
</tr>
</tbody>
</table>
Subscription Matching

• bit-matrix based
  subscription IDs
  
  \[
  \begin{array}{c}
  \text{pred. bit vec} \\
  1 \\
  0 \\
  1 \\
  1 \\
  0 \\
  \end{array}
  \]
  
  hit vector
  
  preds-per-sub

• list based

  \[
  \begin{array}{c}
  \text{pred. bit vec} \\
  s17 \\
  s11 \\
  s6 \\
  s7 \\
  \end{array}
  \rightarrow
  \begin{array}{c}
  \text{hit vector} \\
  s4 \\
  s3 \\
  s1 \\
  s9 \\
  \end{array}
  \rightarrow
  \begin{array}{c}
  \text{preds-per-sub} \\
  s13 \\
  \end{array}
  \]

  + sub-pred association to support deletion
Clustering Algorithm

Indexes

A

B

C

Access
Predicate vector

S1: \textbf{p1}, \textbf{p2}, \textbf{p4}
S2: \textbf{p1}, \textbf{p3}, \textbf{p5}
S3: \textbf{p1}, \textbf{p4}

List of clusters

\textbf{p1} \rightarrow \textbf{s1}, \textbf{s2} \rightarrow \textbf{s3} \rightarrow 1

Detail of cluster

\textbf{s1}\textbf{s2}

\textbf{p2} \textbf{p3} \textbf{p4} \textbf{p5} \textbf{p6}

E: (A, 5), (B, 6), (C, 3)
satisfies \textbf{p1}, \textbf{p2}, \textbf{p3}, \textbf{p4}

Bit Vector

1 1 1 1 0 0

\textbf{p1} \textbf{p2} \textbf{p3} \textbf{p4} \textbf{p5} \textbf{p6}