Event semantics in asynchronous distributed event middleware

Janak J Parekh

Candidacy Exam

May 5, 2003

Agenda

- Preliminaries
  - Overview and definitions
  - Motivation and problem domains
  - Organization of the talk
- Existing work
  - Classic event systems
  - Single event filtering
  - Event sequence filtering
  - Multiple event correlation
  - Other/commercial approaches
- Unsolved problems
- Conclusion

Overview and definitions

- *Events* are discrete, structured data objects generated at a specific point in time
- Events are commonly used to:
  - Communicate between nodes in a distributed system, and possibly different systems
  - Signal alarms or faults
  - Report system activity

Overview and definitions (II)

- Event mechanisms: Given events, we…
  - Propagate/replicate (across objects, processes, network…)
  - Store (later playback, postmortem analysis)
  - Filter (compression, conversion, etc.)
  - Correlate (root-cause analysis)
- This talk deals with the latter 3, *not* the first

Overview and definitions (III)

- We examine two primary types of systems
- Event publish/subscribe middleware
  - Defines event types or formats
  - Construct, publish, deliver and filter events
- Event correlation and event-based workflow
  - Advanced filtration/compression, root-cause analysis, postmortem analysis, etc.
- Surprisingly minimal crossover

Motivations and problem domains

- As the number of nodes increases, the number of events generated among them rises superlinearly [8, 9]
- At the same time, need to gain greater insight into events' semantics
- Various applications
  - Workflow optimization
  - Network management
  - Anomaly, fault detection: EMERALD [9], KX

Organization of the talk

- "Levels" of semantic power
- Publish-subscribe systems
  - Classic event systems: Channel-based with minimal event filtering
  - Single event filtering: Provides flexible filtering on an event-by-event basis
  - Event sequence filtering: Supports delivery and manipulation of contiguous sequences of events
- Multiple event correlation: Dedicated correlation engines that provide flexible filtering and aggregation over many events

7

Agenda

- Preliminaries
  - Overview and definitions
  - Motivation and problem domains
  - Organization of the talk
- Existing work
  - Classic event systems
  - Single event filtering
  - Event sequence filtering
  - Multiple event correlation
  - Other/commercial approaches
- Unsolved problems
- Conclusion

8

Classic event systems

- In essence, event-structured multicast
  - Sources publish to a channel in the middleware layer
  - All subscribers to (clients of) that channel receive the notification
  - Variations include having selection based on single subject/topic
- Events are usually formatted as primitive or opaque structures

9

Classic event systems (II)

- Field [18], 1990
  - One of the first "standalone" event systems, built for GUI component collaboration
  - Events were just strings; they were parsed for simple equality matching (scanf-like syntax)
- CORBA Event Service [27], 1993-2000
  - Events are either generic (opaque) or typed (IDL)
  - No filtering (effectively multicast)

10

Classic event systems (III)

- TIB [24], 1993
  - Objects (LISP CLOS-style)
  - Subject field, equality and wildcard matching
  - Basic subject rewriting
- Others
  - System logs (UNIX, NT)

11

Single event filtering

- CORBA Notification Service [28]
  - Extension to Event Service to solve limitations
  - In particular, "structured" events: IDL-typed events were too hard; enables Boolean-expression filters
  - Many commercial implementations
- JECho [21]
  - Serialized Java objects
  - "Eager handlers" filter information beforehand
- Elvin [17]
  - Structured events similar to CORBA
  - Filter language supports equality, boolean, regular-expression operation

12

Single event filtering (II)

- Siena [23]
  - While sequences are theoretically supported in the interface, unsupported at this point
  - Structured events; XML subsets parsed into attribute-value pairs
- JMS [20]
  - CORBA-like, except no wire format
  - Channel-based; filters on Properties in header; several types of opaque bodies
- WebFilter [25]
  - Notably one of the few XML-capable filter tools
  - But only grabs XPath-specified XML subsets, like Siena, and uses Le Subscribe for underlying work

13

Event sequence filtering

- TAO RT events [16]
  - Built on top of CORBA Notifications; supports simple sequence matching (event batching)
- Gryphon
  - Information flow operations [19]: collapse, expand
  - Stream interpretations [12]: map “equivalent” streams
- READY [26]
  - Supports a type hierarchy of attribute-value structured events (simple concatenation)
  - Boolean expressions can be matched against an array of events, using SQL-like WHERE clauses

14

Multiple event correlation

- Conceptual Framework [3] for network management-based event correlation and filtering
  - Goal is usually RCA or compression
  - Geared towards Managed Objects (MOs), but generally applicable
  - Causal vs. temporal
- Action-Oriented Analysis [10] provides somewhat alternate view
  - Need to determine *actionable* events in order to conduct repair/reconfiguration

15

Multiple event correlation (II) Matching strategies

- FSM/Petri nets
  - Dependency graphs [2]
    - Simple, integrate with existing event management system
- Rule engines
  - Yemanja [1]
    - Layered rule engine for network management
  - WEC, built on top of CORBA [15]
    - Upgrade CORBA events for filtering, sequencing, ECA rules

16

Multiple event correlation (III) More matching strategies

- Procedural languages
  - Rapide [4]
    - “Event Pattern Language (EPL)” – very powerful, procedural
    - Supports both temporal and causal structures
- Signature/Codebook
  - SMARTS InCharge/DECS [6, 8, 11]
    - Actually compiled down from higher-level language
    - Very, very fast, but no temporal constraints

17

Multiple event correlation (IV) Manual rule generation

- An Oracle (i.e., human being)
  - Code rules, build state machines
  - Error-prone and slow
- Rapide [5]
  - Software architecture, architectural constraint languages as model
- Database queries via EvE [13]
  - Build a query engine for event workflow logs

18

Multiple event correlation (V) Automated rule generation

- MODEL [6, 8]
  - Bayesian (probabilistic) learning models from higher-level domain languages
- MEDD [7]
  - Rules from event logs accomplished via systematic search
- Process mining [14]
  - Manual workflow design doesn't work
  - Uses frequency tables

19

Other / Commercial approaches

- Fault management (classic)
  - IMPACT: End-users build expert systems
  - ECXPERT: Correlation trees
- Commercial
  - IBM Tivoli
  - HP OpenView: Circuit-based approach
  - NetCool MicroMUSE
  - TIBCO, Vitria: evolution of TIB

20

Agenda

- Preliminaries
  - Overview and definitions
  - Motivation and problem domains
  - Organization of the talk
- Existing work
  - Classic event systems
  - Single event filtering
  - Event sequence filtering
  - Multiple event correlation
  - Other/commercial approaches
- Unsolved problems
- Conclusion

21

Unsolved problems

- Pushing event correlation into the publish-subscribe layer
  - Some primitive operations are now supported, e.g., Gryphon, READY, but very little higher-level constructs
  - Support a variety of applications, such as mobile/disconnected behavior, seamlessly
  - Better temporal support (distributed clocks?)
  - Implement in existing or new pub-sub event systems

22

Unsolved problems (II)

- Complex type matching and correlation
  - Few systems handle even single-event XML matching
  - Leverage semantics known about sequences
  - Use dynamic type system
    - XML (Schema), PSL SmartEvents (SOAP [29])
    - Global types that no one wanted to do
- Rule generation
  - Learning and searching exist - but other heuristics?
  - Architectural description languages for application and system management

23

Unsolved problems (III)

- Novel correlation domains
  - GUIs
  - Model existing protocols (TCP, syscalls, NFS) as events to reap benefits of correlation
- Learning domains
  - Unsolvable?
  - Or maybe one can teach the system interactively?

24

Conclusion

- The event construct provides a flexible, broadly available methodology for data interchange, fault communication, and history
- Numerous systems already exist to process single and multiple events, but many disconnects and manual operation
- Open field ripe for further study

25

That's all, folks!

(Figures follow… use hyperlinks from earlier slides.)

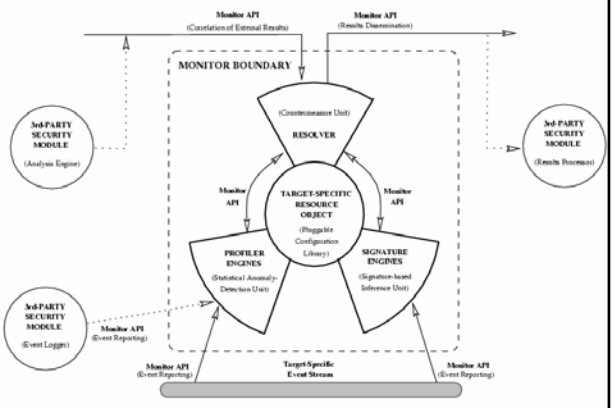
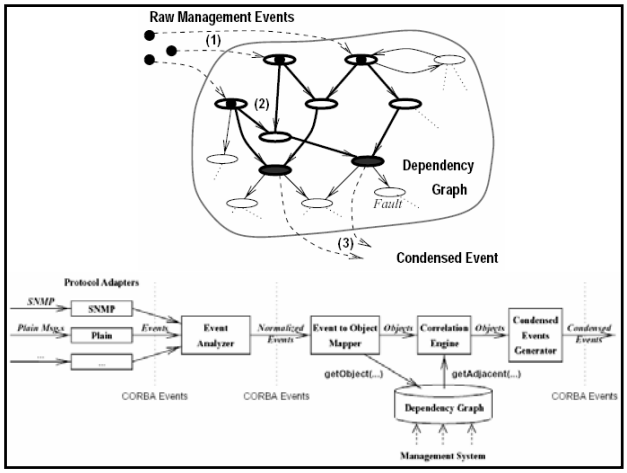


Fig. 1. The Generic EMERALD Monitor Architecture



Rapide example

```
type Table(numPhils : integer) is interface
    action in  StickRequested(n : Chopstick; id: PhilosopherId),
               StickRecovered(n : Chopstick);
    action out ReleaseStick(n : Chopstick; id: PhilosopherId);
behavior
    action  FreeStick(n : Chopstick);
begin
    start => for i: integer in 0..(numPhils-1) do      -- table rule 1.
               FreeStick(i);
             end for;;

    (?a in Chopstick, ?i in PhilosopherId)             -- table rule 2.
     StickRequested(?a, ?i) and FreeStick(?a) |> ReleaseStick(?a, ?i);;

    (?a in ChopStick)                                  -- table rule 3.
     StickRecovered(?a) |>  FreeStick(?a);;
end Table;
```

--

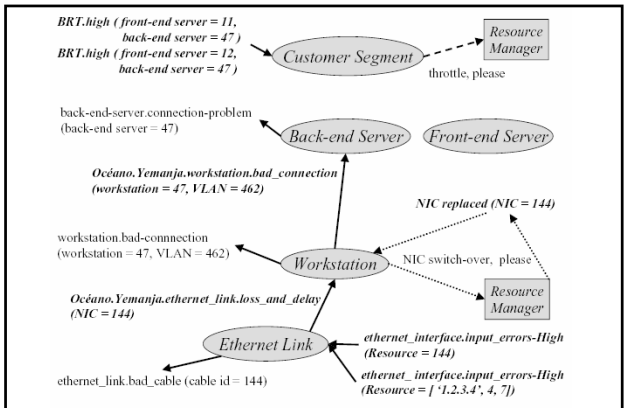


Figure 4: A correlation scenario for `bad_cable` condition

30

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
P_2	0	1	1	0	0	1	0	0	1	1	0	1	1	0	0	1	1	1	1	0
P_2	0	1	0	1	1	1	1	0	0	1	0	0	0	0	1	1	0	1	1	0
P_4	1	1	1	0	1	0	0	1	0	1	1	1	1	0	0	0	1	1	0	0
P_5	0	0	0	1	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	1
P_6	1	0	0	0	0	1	0	1	1	0	0	0	1	1	1	1	0	0	0	1

Table 4: Correlation Matrix

	1	3	4	6	9	18
P_1	1	1	1	1	0	0
P_2	0	1	0	1	1	1
P_2	0	0	1	1	0	1
P_4	1	1	0	0	0	1
P_5	0	0	1	0	1	0
P_6	1	0	0	1	1	0

Table 6: Codebook of Radius 1.5

31

```
interface TransportConn
{
      propagate symptom PacketLossHigh =
            Port, ConnectedTo, PacketLossHigh;
}

interface UDPPort: Port
{
      propagate symptom PacketLossHigh =
            Appl, Underlying, PacketLossHigh;
}

interface MM_InPort: Appl
{
  instrumented attribute long MinRate;
  instrumented attribute long MaxRate;
  instrumented attribute long MsgCounter;
  instrumented attribute long ActTime;

  computed attribute ActualRate = (MsgCounter)/(_time - ActTime);

  event BadRate = (MinRate > ActualRate)||(ActualRate > MaxRate);

  problem PacketLossHigh = BadRate 1.0;
}
```

	#B	B<A	A>B	B<<<A	A>>>B	A→B
T10	1035	0	581	348	1035	0.803
T5	3949	80	168	677	897	0.267
T11	1994	0	0	528	1035	0.193
T13	1000	0	0	0	687	0.162
T9	1955	50	46	366	538	0.161
T8	1994	68	31	560	925	0.119
T3	3949	146	209	831	808	0.019
T6	1035	0	0	348	348	0.000
T7	959	0	0	264	241	-0.011
T12	994	0	0	528	505	-0.093
T1	1000	0	0	687	0	-0.246
T2	1994	0	0	1035	505	-0.487
T4	1994	691	0	1035	505	-0.825

Table 1: an example D/F-table for task *T6*.

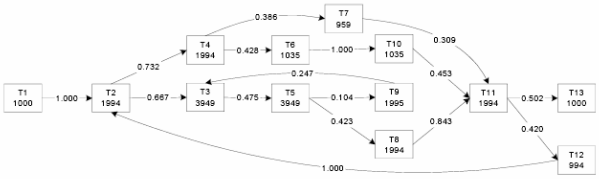


Figure 2: the automatically mined D/F-graph.