A Framework for Event Composition in Distributed Systems

Christian Hälg,
15.6.2004

By Peter R. Pietzuch, Brian Shand, and Jean Bacon

Overview

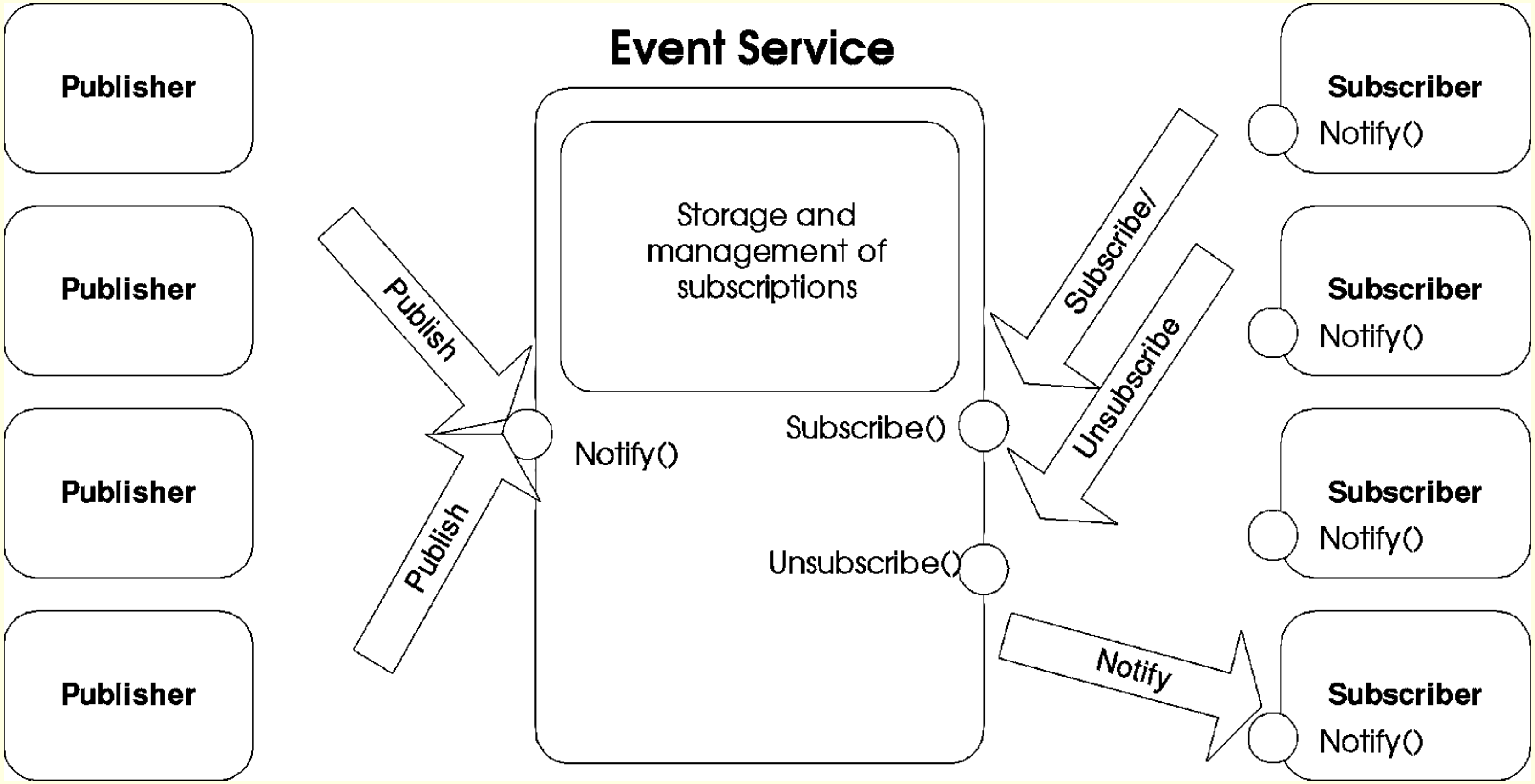
- Introduction
  - Events and Pub/Sub systems
  - Example
  - Motivation
- Composite Event Detection
  - Model and composition language
  - Mobile Composite Event Detectors
  - Policies
- Conclusion

Introduction

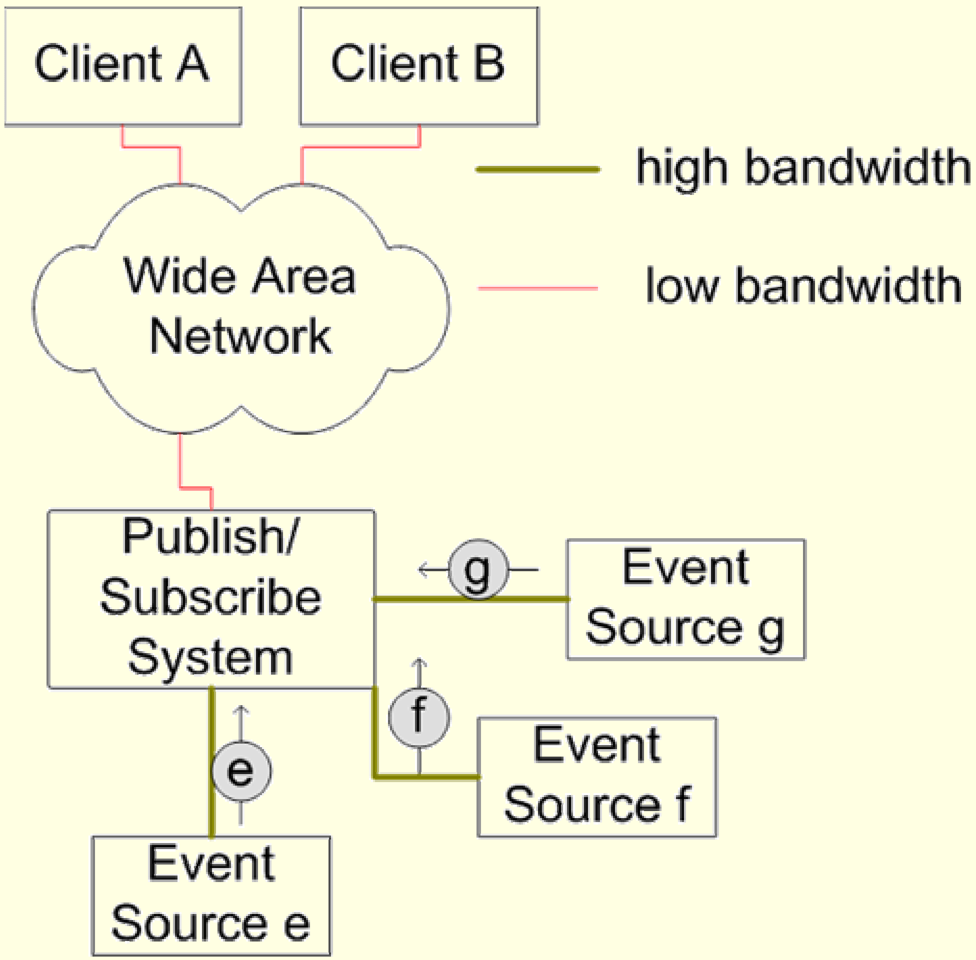
- Events
  - Something has happend
  - Modelled as an object
- Publish/Subscribe Systems
  - „Proxy“ for events
  - Decoupling space, time and synchronization
  - JMS, JORAM
  - Gryphon (IBM)
    - used at Sydney Olympics, US Tennis Open,…

Introduction (2)

- Interface of a Pub/Sub System:



Example – Pub/Sub System

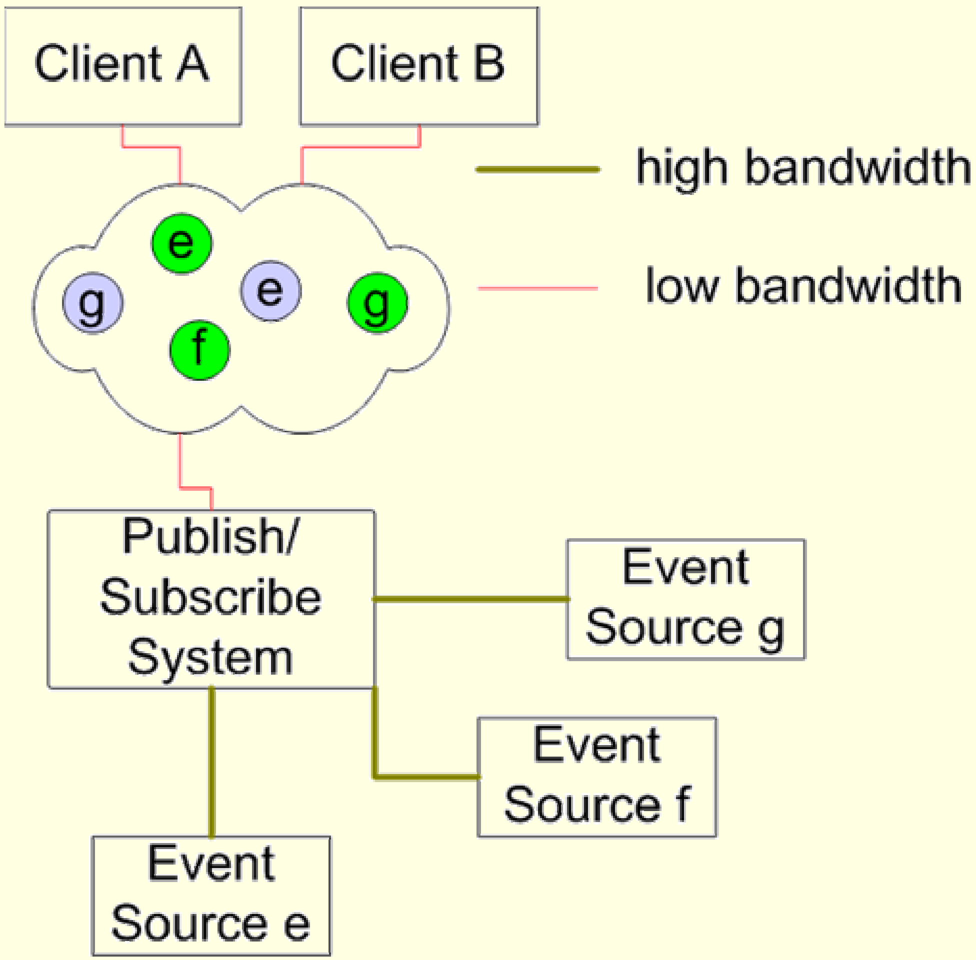


Sources: produce events of type e,f and g.

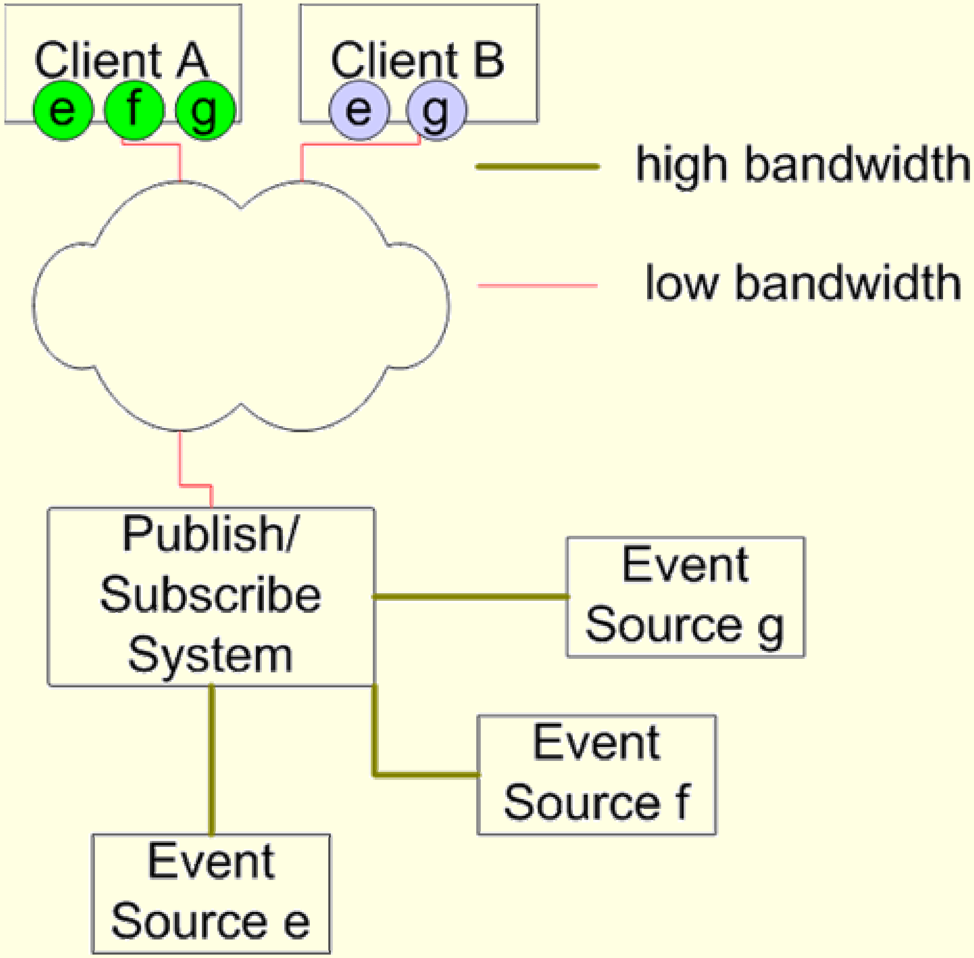
Client A: subscribed for events of type e,f,g and is interested to know, whether they occured in a sequence c1 = (e ; f ; g)

Client B: subscribed for events of type e and g and is interested to know, whether they occured within a 5 minute time interval. c2 = $(e,g)_{T=5min}$

Example – Pub/Sub System (2)



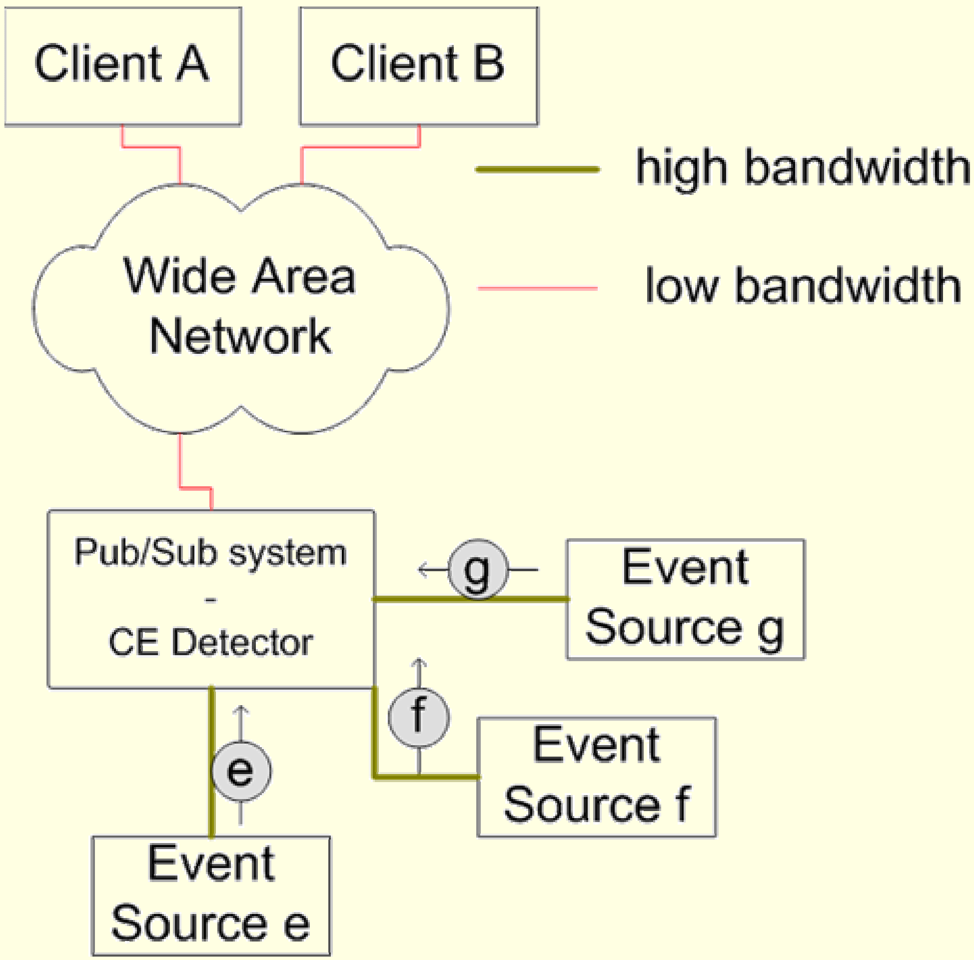
Example – Pub/Sub System (3)



A lot of events are sent to the clients over low bandwidth connections.

Detection of patterns left to the client applications

Example - Pub/Sub System and CE detector (4)

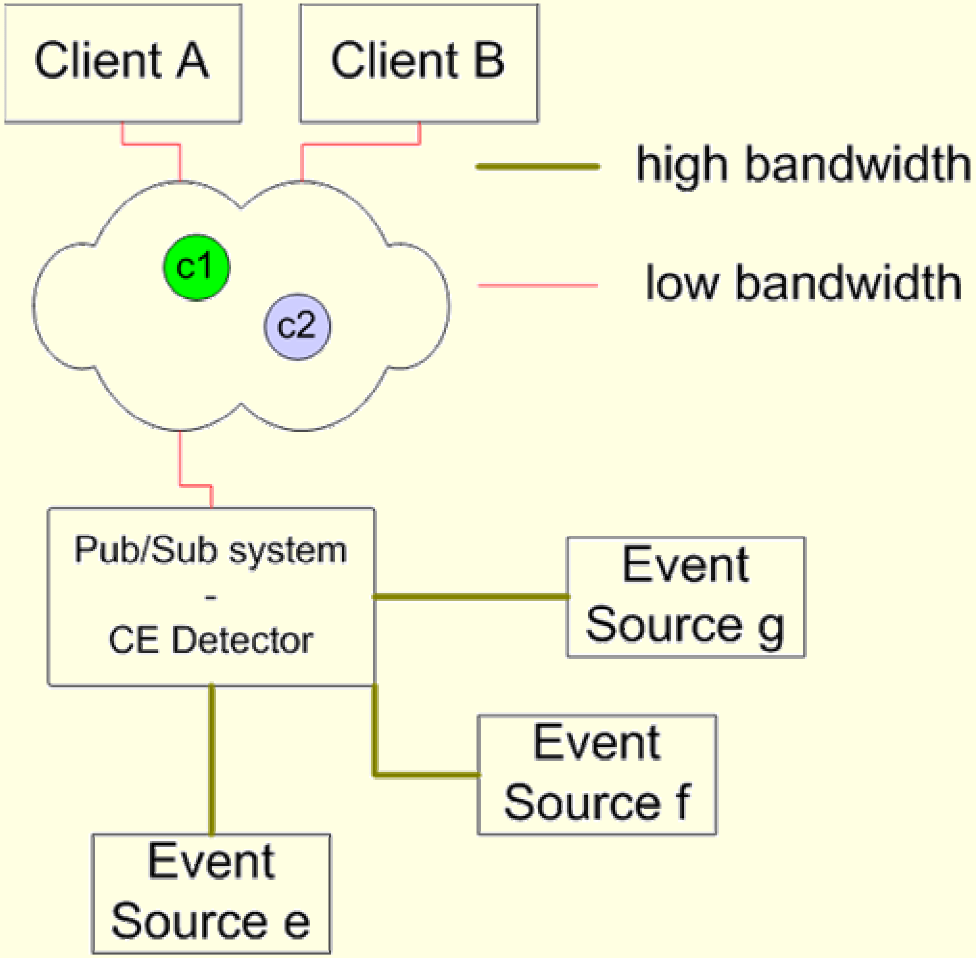


Pub/Sub system is extended with a Composite Event (CE) Detector

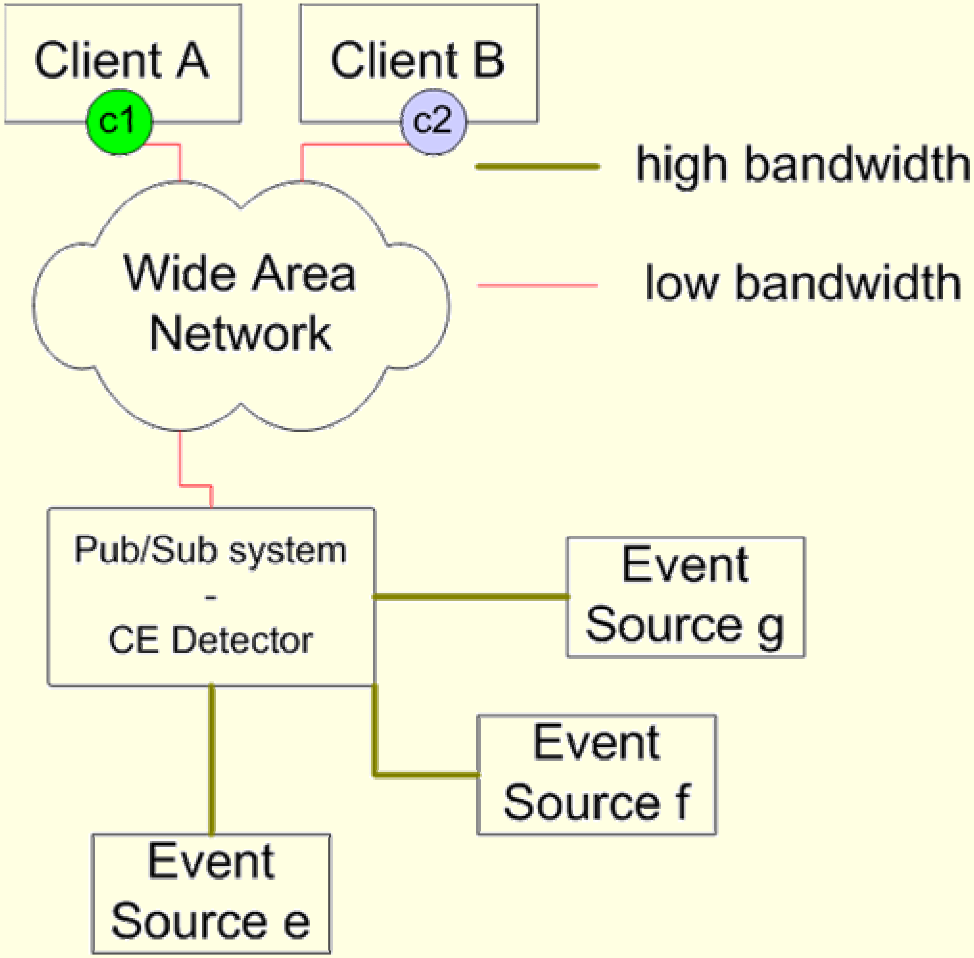
Client A: subscribes for CE c1 = (e ; f ; g) (Sequence)

Client B: subscribes for CE c2 = $(e, g)_{T=5min}$

Example - Pub/Sub System and CE detector (5)



Example - Pub/Sub System and CE detector (6)



Less traffic between Pub/Sub System and the clients.

Client doesn‘t have to care about detection of (probably) complex event-patterns.

Motivation

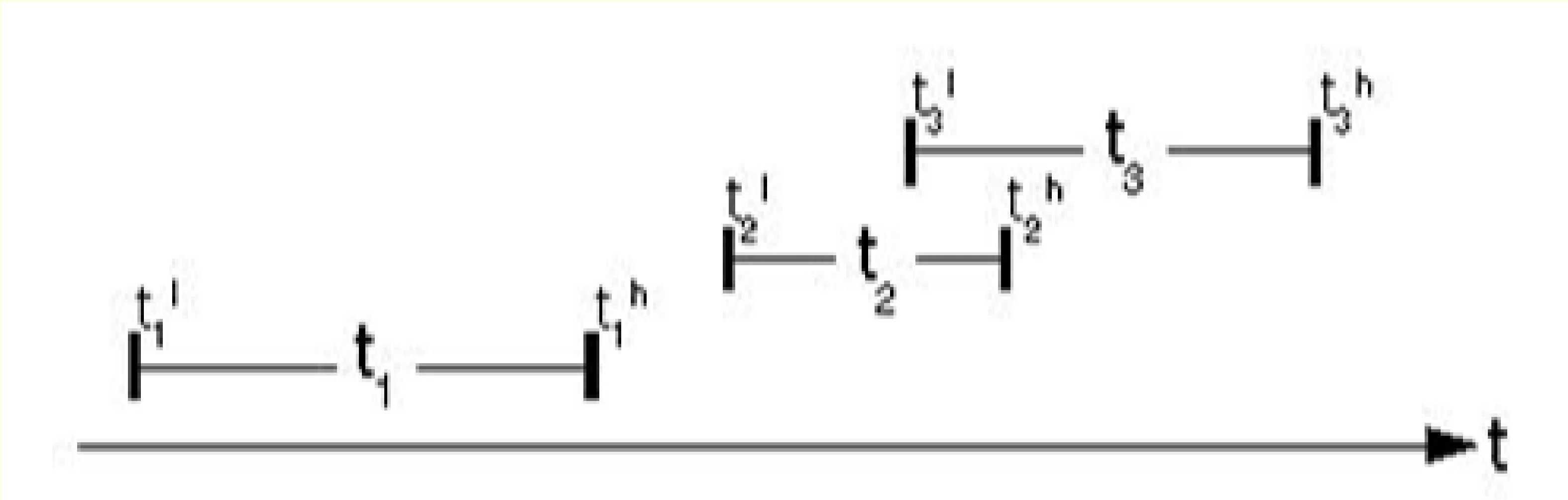
- Abstraction of Events
  - Composition of atomic events
  - CE represented as a new atomic event
  - Composition of CE
- Reduction of bandwidth usage

Composite Event (CE) Detection

- Result: CE Detection Framework
  - Regular language for CE specification
  - Compiler translates the „core CE language“ into detectors
  - Realised as finite automata
- Assumptions on infrastructure
  1. An underlying Pub/Sub System
  2. Events carry a timestamp, denoted by an time interval

Composite Event (CE) Detection (2)

Time Model:

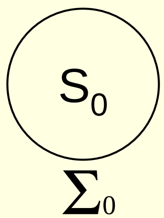


„strong" order: $t_{high}^{A} < t_{low}^{B}$ $\quad (t1 < t2) \wedge (t1 < t3)$

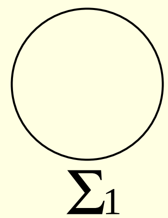
„weak" order: $\left(t_{high}^{A} < t_{high}^{B}\right) \vee \left(t_{high}^{A} = t_{high}^{B} \wedge t_{low}^{A} < t_{low}^{B}\right)$ $\quad t1 \prec t2 \prec t3$

Composite Event (CE) Detection (3)

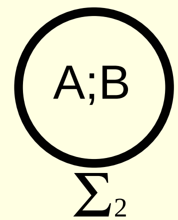
Inital State



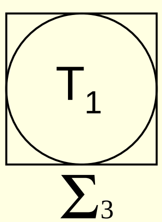
Ordinary State



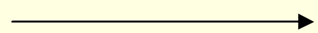
Generative State



Generative Time State

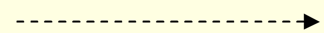


strong order transition



($<$)

weak order transition



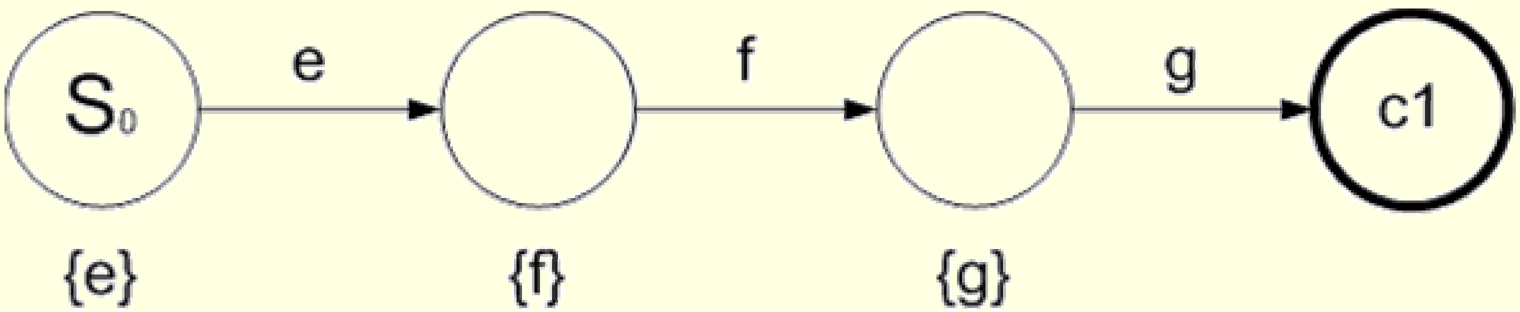
($\prec$)

Composite Event (CE) Detection (4)

Continue example:

Client A: c1 = (e;f;g)   (sequence of 3 events)

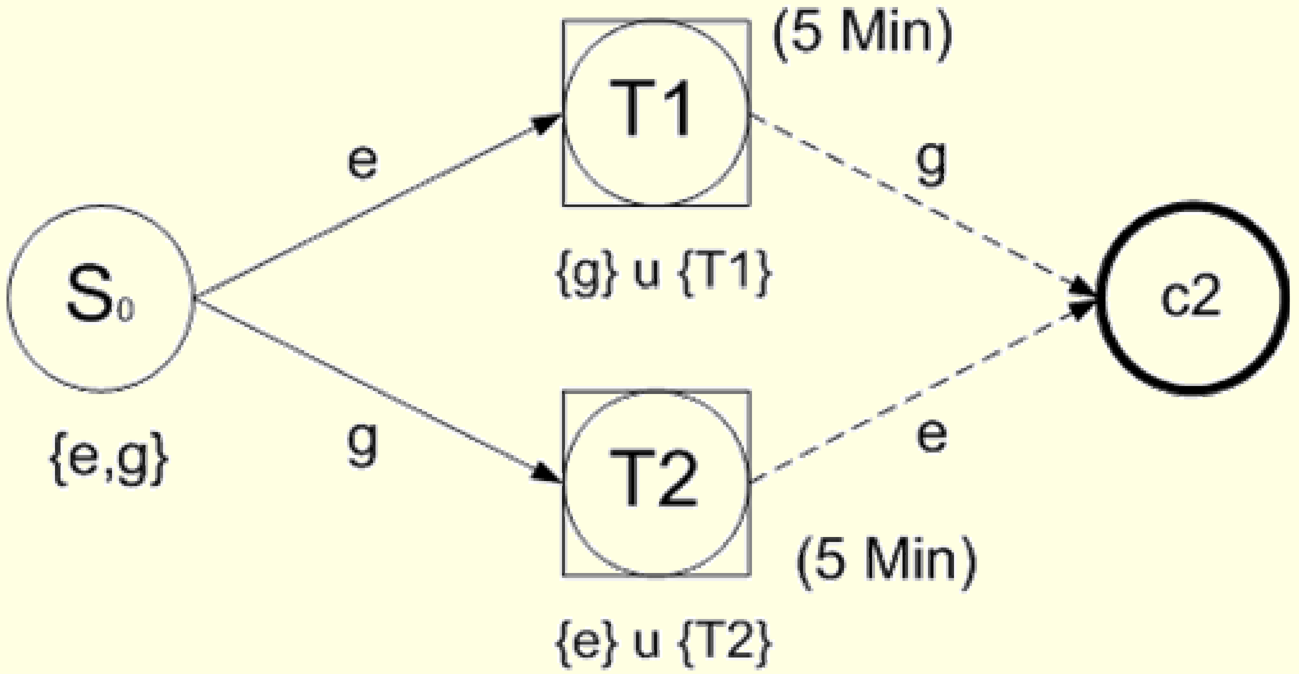
Transformation into automaton:



Composite Event (CE) Detection (5)

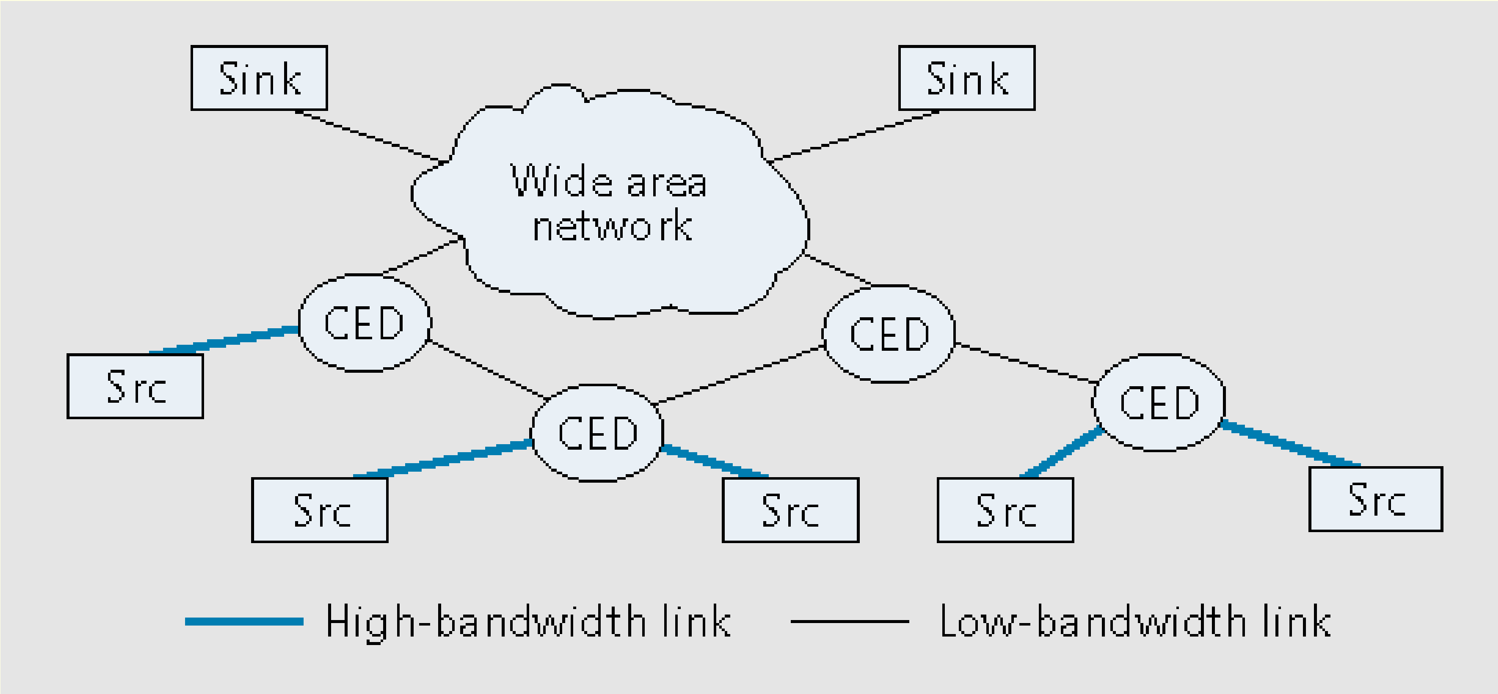
Client B: $c2 = (e, g)_{T=5min}$

Transformation into automaton:



Distributed Detection

- In the example: monolithic system for event detection
- Distribute detectors over the network



Mobile Composite Event Detectors

A Mobile Detector can…

factorize	CE expressions
instantiate new detectors	to reduce own load
migrate to another node	to reduce bandwidth usage at bottlenecks
destroy itself	if no longer needed

- Detectors move to „optimal“ site
- Tradeoff between bandwidth usage and latency

Distribution Policy

1. Determine location of a detector.
2. Decide on the degree of decomposition and distribution.
3. Decide on duplication of certain (often used) detectors.

- Optimization in terms of bandwidth usage and latency.

Detection Policy

- When can an input stream of events safely be processed?
- **Best Effort Detection**: events in input stream are consumed without delay.
- **Guaranteed Detection**: consumption of an event in the input stream, if event is *stable*.
  - *stable*: An event is stable, if an event of the same source arrived with a later timestamp.

Detection Policy (2) – Problems

- What if a source is rarely publishing events?
  - Publish dummy heartbeat events.
  - Load on bandwidth.
- What if a source is disconnected?

- Research is done on a probabilistic stability measure.

Conclusion

- System tested on artifical office example
  - Bandwidth usage can be brought down significantly
  - But no „real world“ test yet
- Paper is sloppy concerning the formal definitions
- Composite event detection with FSMs isn‘t really new (already done for database trigger events)

Conclusion (2)

- Contributions
  - Idea of detection of composite events applied to distributed systems
  - Notion of mobile, decomposable and clonable detectors

Thank You

Questions?

Appendix A – Time Model

- Events have a timestamp denoted by a time interval t = [t_{low},t_{high}]

- Partial Order:
  - Time intervals of event A and B do not overlap
  - A < B iff $t_{high}^{A} < t_{low}^{B}$
  - strong order
  - denoted by a solid arrow in the FSM

Appendix A – Time Model (2)

- Total Order
  - Time intervals do overlap
  - $A \prec B$ iff $\left(t_{high}^{A} < t_{high}^{B}\right) \vee \left(t_{high}^{A} = t_{high}^{B} \wedge t_{low}^{A} < t_{low}^{B}\right)$
  - weak order
  - denoted by a dashed arrow in the FSM

Appendix B – Core CE Language

Atoms: Events {A,B,C,…} in input stream

Negation: $[\neg E \subseteq \Sigma] \equiv [\Sigma \setminus E \subseteq \Sigma]$

Concatenation: C_1C_2 (C_1 weakly followd by C_2)

Sequence: $C_1;C_2$ (C_1 strongly followed by C_2)

Iteration: C_1*

Alternation: $C_1|C_2$

Timing: $(C_1,C_2)_{T\text{ = timespec.}}$

Parallelisation: $C_1||C_2$

Appendix B – Core CE Language (2)

Brian enters the room followed by Peter

$[B];[P]$

Brian enters the room before Peter

$[B \subseteq \{B, P\}]$

Brian enters and Peter follows within an hour

$([B],[P \subseteq \{P, T1\}])_{T1=1h}$

Someone else enters the room when Brian is away

$[B][\neg B \subseteq A][B]$

Appendix C - Implementation

- Framework built on top of JORAM (Java Open Reliable Asynchronous Messaging), which is a open-source implementation of JMS (Java Message Service).
- Download at: http://joram.objectweb.org/
- Source of the framework not yet available for download (but probably soon)