Scalable High-Performance Event Filtering for Dynamic Multi-point Applications

Douglas C. Schmidt
schmidt@cs.wustl.edu

Washington University, St. Louis

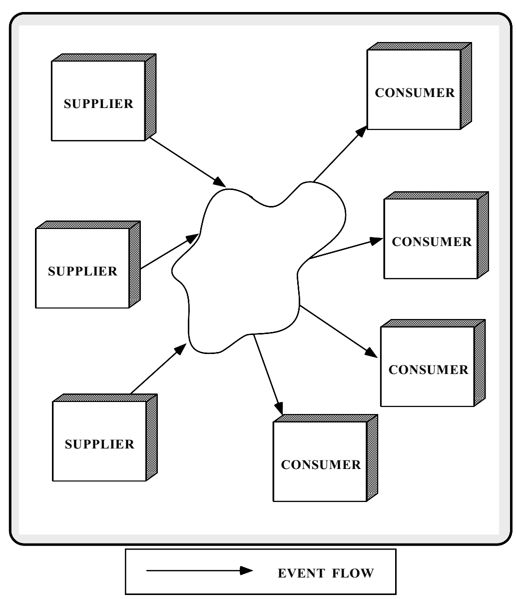
Introduction

- *Dynamic multi-point ("DMP")* applications benefit from high-performance *event filtering*
- DMP applications include:
  - *Satellite telemetry processing*
  - *Large-scale network management*
  - *Real-time market data analysis*
  - *On-line news services*
  - *Distributed agents in mobile personal communication systems*

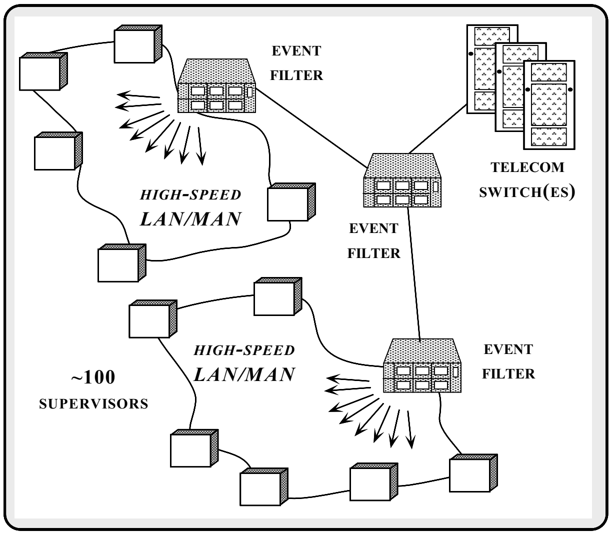
Key Characteristics of Dynamic Multi-point Applications

- *Multiple suppliers*
  - Continuous stream of events (messages) generated in real-time
  - Potentially complex event data formats
  - High volume of events
- *Multiple consumers*
  - Consumers process events in real-time
  - Each consumer may subscribe to a subset of total events
  - Consumers may add, delete, or modify subscriptions dynamically

General Structure of Dynamic Multi-point Applications



Ericsson DMP Applications



- Call center management

Event Filtering Overview

- In DMP applications, each event sent by a supplier may be sent to a different subset of consumers
  - *i.e.*, not all consumers receive every event
- Thus, event filtering is an important "data reduction" technique
- Filtering may occur at multiple locations in a distributed system

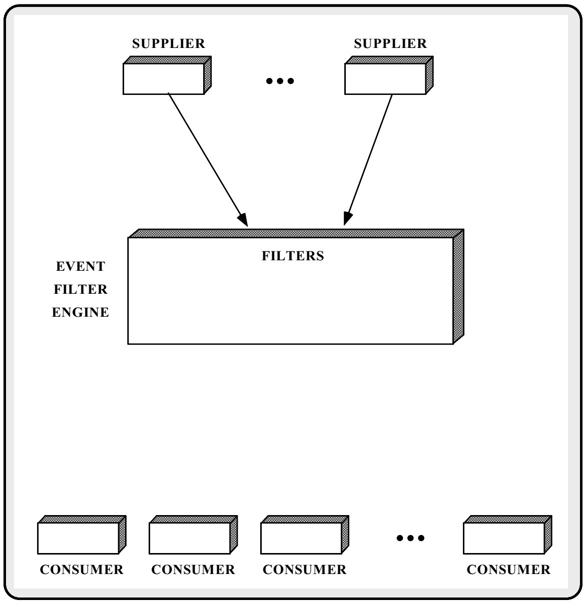
Event Filtering Criteria

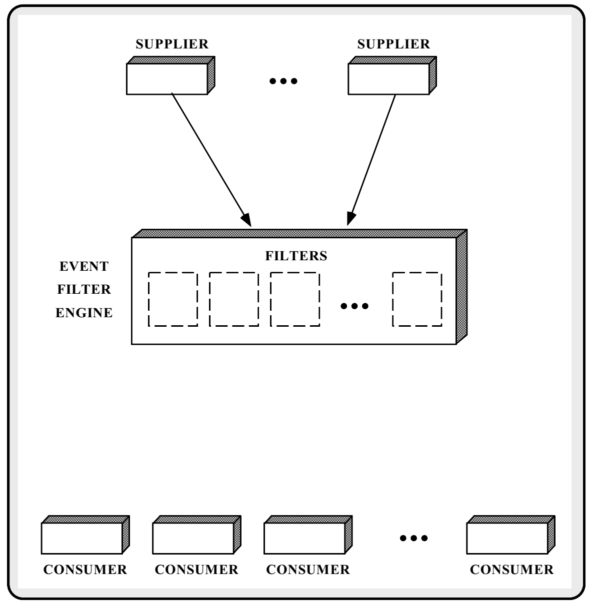
- Stateless
  - *Event type*
  - *Event values*
  - *Event generation time*
- State-based
  - *Event frequency*
  - *Event state changes*

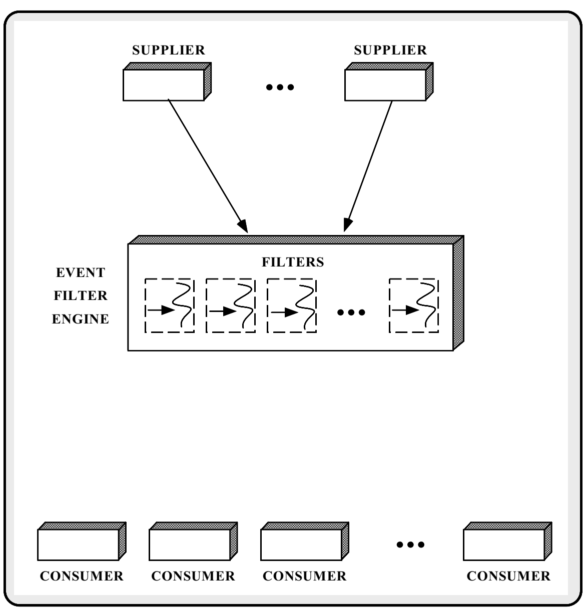
Research Topics

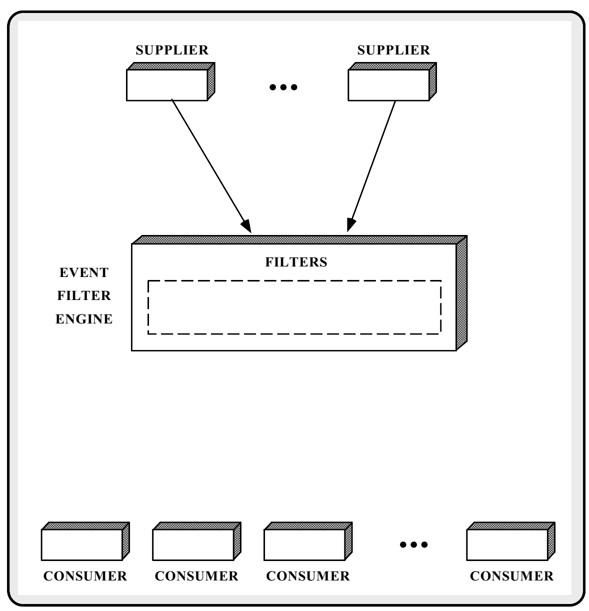
- *Performance*
  - High throughput and low delay
  - Load balancing
  - Scalability
- *Flexible/lightweight Configuration*
  - Automated type checking, generation, and optimization of filters
  - Flexible partitioning and placement
  - Dynamic extensibility

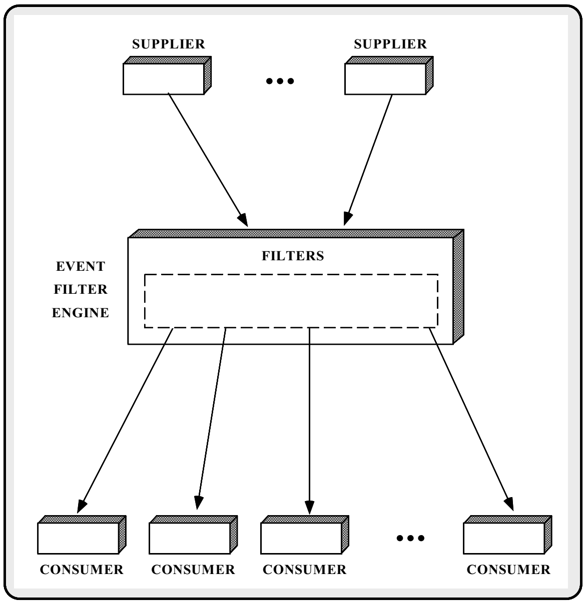
Scalability Optimizations

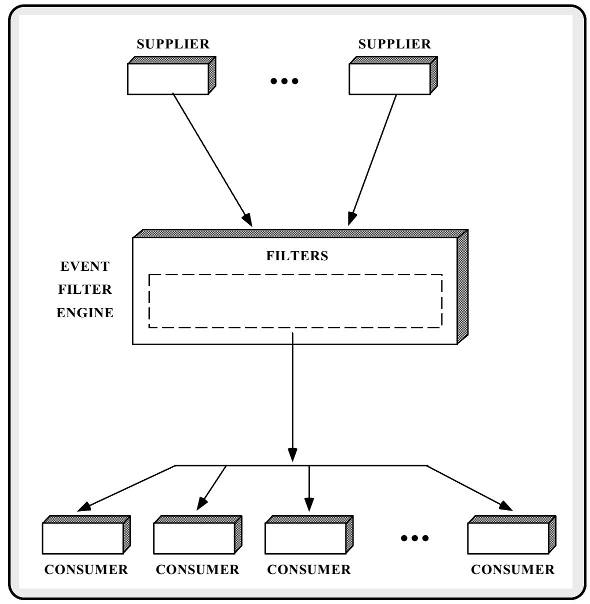












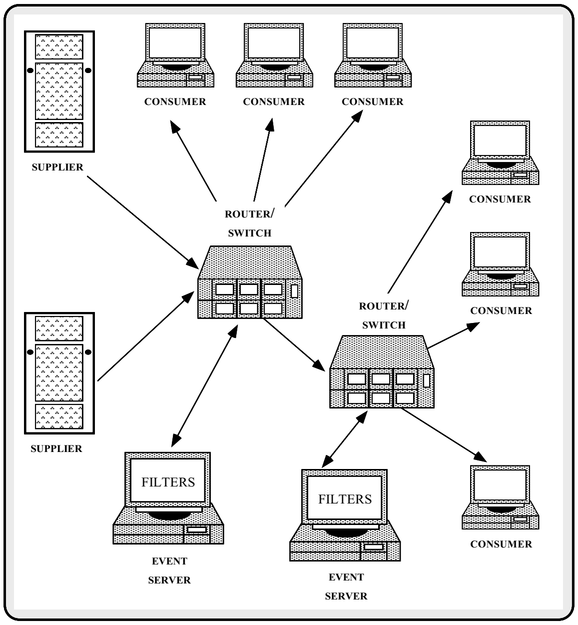
Optimization Techniques

- Event filtering optimizations:
  - Compile rather than interpret
  - Coalesce multiple filters
    - *Dynamic trie*
    - *Finite automaton*
  - Process filters in parallel
  - Distribute vs. centralize filters

Flexible/Lightweight Filter Configuration

- Filters generated and optimized automatically
  - Based on OMG IDL and ACE OO framework
- Partitioning and placement of event filters may be deferred until installation-time or run-time
- Explicit dynamic linking provides lightweight extensibility at run-time
  - Facilitates *compilation*, rather than *interpretation*

Distributed Event Filtering



Filter Schema Notation

- Schema notation is based on a superset of OMG IDL
- Properties of OMG IDL
  - Implicitly typed
  - Supports complex data types
- Extensions to IDL for event filtering
  - Bit fields
  - Meta-data

Example Filter Schema

- Format of a logging record defined in OMG IDL

```
module Logger
{
  // Types of logging messages.
  enum Log_Priority {
      LOG_DEBUG,   // Debugging messages
      LOG_WARNING, // Warning messages
      LOG_ERROR,   // Errors
      LOG_EMERG    // A panic condition
  };

  // Format of the logging record.
  struct Log_Record {
    Log_Priority type;
    long         time;
    long         app_id;
    long         host_id;
    sequence<char> msg_data;
  };
};
```


Filter Expression Language

- Based on superset of C++ expressions
  - "N in a row"

```
// Matches if 10 consecutive error messages sent from
// an application with a particular host and app id.

"this->app_id == 2001 && this->host_id == x7237d923
 && this->type{0..9} == Logger::LOG_ERROR"
```

  - "State changes and thresholding"

```
// Matches if the absolute value of the length of
// two consecutive messages from application 2010
// differ by more than 100 bytes.
"this->app_id == 2010 && abs (this->length{0} -
                              this->length{1}) > 100"
```

  - *Timestamps*

```
// Matches if the time stamp of the message is
// between noon and 1 pm.
"this->time >= 12:00:00 && this->time <= 13:00:00"
```

Related Work

- ISIS News
  * Filtering at destination only
  * Simple filtering criteria (*i.e.*, character strings)
- Packet filters
  * Primarily interpreted, not compiled
  * Limited support for generalized coalescing
  * Limitations on filtering criteria
- HP OpenView OSI event services
  * Interpreted
  * Exceedingly inefficient process architecture
- OMG CORBA Services
  * Defines an event filter constraint language

Work in Progress

- Evolve OO framework prototype
  - Based on OMG CORBA
- Integrate the OO framework into testbed environment at Washington University
  - *e.g.*, ATM networks and 20-CPUs SPARCcenter 2000 parallel processor
- Use OO framework and testbed to conduct experiments that identify and alleviate bottlenecks in dynamic multi-point applications
  - Event traffic patterns based on production DMP applications

Concluding Remarks

- A growing class of distributed applications require support for high-performance, distributed event filtering
- Extensible object-oriented framework for event filtering helps to:
  1. Simplify application development, configuration, and reconfiguration
  2. Enable extensive optimizations
- Wash. U. infrastructure provides high-speed ATM networks and parallel processing to experiment with event filtering for dynamic multi-point applications