Publish/Subscribe
Publish/Subscribe

- **Model**
  - Producers *publish* information
  - Consumers *subscribe* to information
  - Usually producers and consumers both in *push* mode

- **Decoupling of participants**
  - In time
  - In space
  - In flow

- **Enforces scalability**
Topic-Based Publish/Subscribe

- A.k.a. *subject*-based publish/subscribe

- News-like approach
  - Messages are classified according to topic names, e.g., ETHZ
  - Topics can be seen as (dynamic) groups

- URL-like topic names for convenience
  - Topics arranged in a hierarchy, e.g., \( /ETHZ/CSE \)
  - Automatic subscriptions to subtopics
  - Wildcards
  - Aliases
Topic-Based
Content-Based Publish/Subscribe

- A.k.a. property-based publish/subscribe
- Events classified according to their properties
  - Consumers subscribe by specifying properties of events of interest
  - Application criteria are seen as subscription pattern
  - Translated to filter, or predicate, matched against events

- Classic approach
  - Map event attributes to properties
  - Subscription language and parser,
  - E.g., "name == ‘Bob’"
Content-Based
Self-Describing Messages

- Cf. *DynAny* in CORBA

  ✓ Represent rather *structures* than objects, e.g.,

  ```java
  public class SelfDescribingEvent extends Event {
      public void addString(String fieldName, String s) {...}
      public void addByte(String fieldName, Byte b) {...}
      public void addObject(String fieldName, Object o) {...}
      ...
      public String getString(String fieldName) {...}
      public Byte getByte(String fieldName) {...}
      public Object getObject(String fieldName) {...}
      ...
      public String[] getFieldNames() {...}
      public Class getFieldType(String fieldName) {...}
      ...
  }
  ```
Most topic-based systems nowadays also incorporate content-based features

- More flexible
  - Can be used to express topics

Self-describing messages

- Offer much dynamism
- Enforce interoperability
- Rarely required
- Not type-safe
Type-Based Publish/Subscribe

- Subscription criterion
  - The type (its interface) of application-defined events
  - Content-based queries based on methods

- Combines static and dynamic schemes
  - Static classification should be made as far as possible for efficiency
  - Filters for fine-grained content-based subscription increase expressiveness if required

- Languages which support structural reflection
  - No need for specific events (e.g., Java introspection),
  - In other languages, events can subtype an introspective event type
Type-Based

Event Types

$T_1$

$T_2$

$T_3$

$T_4$

$T_5$

$P_1$

$P_2$

$P_3$
Publish/Subscribe

Java Message Service (JMS)
Java Message Service

- The Java Message Service is only an API
  - Standardized API for messaging in Java
  - Implemented by most industrial solutions
    - TIBCO
    - iBus
    - Gryphon
    - ...

- Two messaging styles:
  - Publish/subscribe (topic-based & content-based): some-of-n
  - Point-to-point (message queuing): one-of-n
Benefit of JMS

- Sun standard
  - Ensures a certain degree of portability
  - Integration with other Java concepts/services
    - *Enterprise Java Beans (EJB)*: asynchronous beans vs. synchronous beans
    - *Java Database Connectivity (JDBC)* for database integration
    - *Java Transaction Service (JTS)* for messages as part of distributed transactions
    - *Java Naming and Directory Intf (JNDI)* for object lookup
  - API can be downloaded: package `javax.jms`
JMS Event Model

- General-purpose messages which require explicit marshalling
- **Message body can contain**
  - ✓ Stream
  - ✓ Properties
  - ✓ String
  - ✓ Object
  - ✓ Bytes
- **Additional attributes**
  - ✓ Message header: explicit messaging
  - ✓ Message properties: for content-based filtering
Message Attributes

- **Message header**
  - Assigned by service upon send
    - Destination
    - Delivery mode (PERSISTENT, NON_PERSISTENT)
    - Message ID
    - Timestamp
    - Priority
    - Expiration
  - Provided by client
    - Correlation ID, e.g., refer to other message
    - Type
    - Reply destination

- **Message properties**
  - Name-to-value properties provided by message producer
  - Property types (native Java types)
    - boolean
    - byte
    - short
    - int
    - long
    - float
    - double
    - String
  - Note: attributes mapped to properties, encapsulation…!
Properties for Content-Based

- Properties of messages are assigned explicitly
  - Not `java.util.Properties`

- Subscriber describes required properties
  - `Message selector = filter`
  - Subscription language: message selector is `String`
  - Syntax specified by JMS
  - Must be mapped to service provider’s subscription language syntax

- E.g.,
  "JMSType = 'car' AND color = 'blue' AND weight > 2500"
Common Facilities

- **Destination**
  - Named object (topic, queue) obtained through JNDI: empty interface

- **ConnectionFactory**
  - Obtained through JNDI, used to create Connection to a topic, queue: empty

- **Connection**
  - May require authentication
  - Register ExceptionListener for problem detection
  - Factory for Session

- **Session**
  - Required by client (producer/consumer) to interact with topic, queue
  - Creates MessageProducer (push), MessageConsumer (push/pull)
  - Single threaded. Transaction support, unacknowledged messages, order, ...
public interface Connection {

    public String getClientID() throws JMSException;
    public void setClientID(String ID) throws ...
;

    public void setExceptionListener(ExceptionListener l)
    throws ...
;
    public ExceptionListener getExceptionListener() throws ...
;

    public void close() throws ...
;
    public start() throws ...
;
    public stop() throws ...
;
...

    /* (Sessions created through implementation classes) */
}
public interface Session {

    public void setMessageListener(MessageListener l) throws ...
    public MessageListener getMessageListener() throws ...

    public TextMessage createTextMessage() throws ...
    public StreamMessage createStreamMessage() throws ...

    ...

    public void close() throws ...
    public void recover() throws ...
    public void commit() throws ...
    public void rollback() throws ...

    ...
}
Message Producers

public interface MessageProducer {

    public void setDeliveryMode(int deliveryMode) throws ...;
    public int getDeliveryMode() throws ...;

    public void setPriority(int defaultPriority) throws ...;
    public int getPriority() throws ...;

    public void setTimeToLive(long ttl) throws ...;
    public long getTimeToLive() throws ...;

    ...

}
public interface MessageConsumer {

    /* Provide content-based filter */
    public String getMessageSelector() throws ...;

    /* Push model */
    public void setMessageListener(MessageListener l) throws ...;
    public MessageListener getMessageListener() throws ...;

    /* Poll */
    public Message receive() throws ...;
    /* Blocking pull */
    public Message receive(long timeout) throws ...;

    ...
}
Point-To-Point (PTP)

**Objects**

- Queue represents a vendor-specific implementation
- TemporaryQueue is a temporary incarnation, bound to a QueueConnection
- Created through a QueueConnectionFactory
- QueueSession, QueueReceiver (message consumer: push/pull), QueueSender (message producer)
- QueueBrowser to query queue without removing messages

... 

**Note**

- Message selector can be specified by consumer
public interface Queue {

    public String getQueueName() throws ...;
    public String toString() throws ...;
}

public interface QueueBrowser {

    public Enumeration getEnumeration() throws ...;
    public String getMessageSelector() throws ...;

    public String getQueue() throws ...;
    ...
}

Publish/Subscribe

- **Objects**
  - **Topic** gives access to pub/sub system: no naming conventions
  - **TemporaryTopic, TopicConnectionFactory, TopicConnection, TopicSession**, as seen previously
  - **TopicSubscriber** *(message consumer)* and **TopicPublisher** *(producer)*

- **Durable subscription**
  - Client provides unique ID

- **TopicRequestor**
  - Use pub/sub to make request/replies

- **Mixed topic/content-based**
  - Client provides a message selector
public interface Topic {

    public String getTopicName() throws ...
    public String toString() throws ...
}

public class TopicRequestor {

    public TopicRequestor(TopicSession session, Topic topic) throws ... {...}
    public Message request(Message message) throws ... {...}

}
JMS Exceptions

- **JMSException**
  - Checked exception
  - Root of exception hierarchy

- **Specific exceptions**
  - **JMSSecurityException**: authentication problem
  - **InvalidDestination**: destination not understood by provider
  - **InvalidSelectorException**: « syntax error » in filter
  - **MessageFormatException**: e.g., unsupported payload class
  ...