Messaging: Basic Exchange, Processing and Transformation Models and Tools

Hong-Linh Truong
Faculty of Informatics, TU Wien

hong-linh.truong@tuwien.ac.at
http://www.infosys.tuwien.ac.at/staff/truong
Twitter: @linhsolar
Outline

- Overview of streaming message-oriented data programming
- Communication - Message-Oriented Middleware
  - Java Messaging Service (JMS), Advanced Message Queuing Protocol (AMQP), Message Queuing Telemetry Transport (MQTT)
- Integration - Enterprise Integration patterns
  - Message routing patterns
  - Message transformation patterns
- Processing - streaming data processing with Complex Event Processing
Topic complexity

Thousand of pages of documents, APIs, tutorials and code

Getting started with each topic of “complex *” in 10 minutes.

Further advanced topics will be covered in Lecture 5
Why messaging is so important for DST?
Overview

STREAMING MESSAGE-ORIENTED PROGRAMMING
Data stream programming

Data stream: a sequence/flow of data units

Data units are defined by applications: a data unit can be data described by a primitive data type or by a complex data type, a serializable object, etc.

Streaming data: produced by (near)realtime data sources as well as (big) static data sources

- Examples of data streams
  - Continuous media (e.g., video for video analytics)
  - Discrete media (e.g., stock market events, twitter events, system monitoring events, notifications)
Some key issues

- Communication
  - Many techniques are needed: sending/receiving, routing, storage, etc.

- Data processing
  - Within the brokering infrastructures and platforms
  - Within the producer and the consumer
  - Interoperability issues: message format, etc.
  - Performance issues: rates/throughput, intervals, delay/latency, processing time etc.
Message-oriented Middleware (MOM)

- Discrete media data units
  - Data units are structured messages (maybe ordered by timestamps)
- Well-supported in large-scale systems for
  - Persistent but asynchronous messages
  - Scalable message handling
- Message communication and transformation
  - publish/subscribe, routing, extraction, enrichment
- Several implementations

- Amazon SQS
- Apache Qpid™
- Apache Kafka
- JMS
- stormmmq
- RabbitMQ
Message-oriented Persistent Communication

Exchange models

Sender running

Sender running

Sender passive

Sender passive

Broker

Receiver running

Receiver passive

Receiver running

Receiver passive

(a) (b) (c) (d)

Operations

PUT/SEND/PUBLISH
GET/RECEIVE
POLL/SUBSCRIBE
NOTIFY/SEND

The receiver pulls the data from the broker or the broker pushes the data to the receiver?

MOM – some message processing operations

Publish/subscribe/notify; send/forward; routing operations within a broker
Message processing within data consumer

Incoming streams

Streaming data type m

\[ m_3 \quad m_2 \quad m_1 \]

\[ \ldots \quad \ldots \quad \ldots \]

Streaming data type s

\[ s_3 \quad s_2 \quad s_1 \]

Application-specific data processing

Output complex messages

Complex/multiple streams data processing
Streaming data processing with a network of data processing elements

Streaming data processing

Streaming data processing

Streaming data processing

...
Message handling for service integration

- Messages handling concepts and patterns have been around for many years
  - Cross services/organizations integration
  - Enterprise integration pattern is well studied but mostly focused on business messages

- Today distributed applications
  - not just enterprise integration patterns
  - also various types of measurements and log information integration
Filter, exchange, etc.

We need several features implemented by MOM, consumers, or external systems

http://www.eaipatterns.com/
Syntax and semantic problems

Source: http://www.smart-words.org/humor-jokes/language-humor/who-is-hu-china.html

- President: "Secretary! Nice to see you. What's happening?"
- Secretary: "Sir, I have the report here about the new leader of China."
- President: "Great. Lay it on me."
- Secretary: "Hu is the new leader of China."
- President: "That's what I want to know."
- Secretary: "That's what I'm telling you."
- President: "That's what I'm asking you. Who is the new leader of China?"
- Secretary: "Yes."
- President: "I mean the fellow's name."
- Secretary: "Hu."
- President: "The guy in China."
- Secretary: "Hu."
- President: "The new leader of China."
- Secretary: "Hu."
- President: "The Chinaman!"
- Secretary: "Hu is leading China."
Message serialization and deserialization

- Remember that the sender and the receiver are diverse
  - In many cases, they are not in the same organization
  - You need to guarantee the message syntax and semantics

- Solutions
  - Agreed in advance → in the implementation or with a standard
  - Know and use tools to deal with syntax differences
  - But semantics are domain/application-specific
Arvo

- https://avro.apache.org/
- Support message description
- Serialize and deserialize libraries
- Work with different languages

Why is it important?
Some other techniques

- **Protobuf**
  - From Google, used by default in gRPC (gRPC.io)
  - [https://github.com/google/protobuf](https://github.com/google/protobuf)
  - Language-neutral, platform-neutral mechanism for serializing/deserializing structured data

- **Thrift**
  - [https://thrift.apache.org](https://thrift.apache.org)
  - RPC style
  - Support also serializing and deserializing data)
  - Support cross-language services development
    - Specify services interfaces
    - Data exchange
    - Code generation

- **Flatbuffers**
  - [https://github.com/google/flatbuffers](https://github.com/google/flatbuffers)
Communication

JAVA MESSAGING SERVICE
General concepts

- **Standard APIs** for Java platform
Message Structure

- **Header**: pre-defined system information (e.g., storage, routing and identification operations)
- **Properties**: application defined properties
- **Body**: application-defined
  - Java primitive types, Map (a set of tuples), Text, Serializable Object

- Types of messages (or what is a message for?)
  - **Application-specific semantics**
    - E.g., notify an event, send a document, or ask for the execution of a command
Delivery Patterns

Point-to-point

Publish/Subscription (Pub/Sub)

Fig source: http://docs.oracle.com/javaee/7/tutorial/doc/jms-concepts002.htm
Request-reply versus Request-only messages

- Request only
  - A sender does not expect a reply for a given request

- Request-reply
  - A sender expects, e.g., a system ack or an application-specific reply

- Some design principles
  - Need to uniquely identify a request message?
    → Use a unique identifier
  - Need a reply message from a request message
    → Where is the return address?
    → Correlation between the request and reply messages (using unique id), e.g., MessageType=REQUEST\|REPLY & MessageID = ID
JMS programming versus administrative activities

Best as administered objects

Best with programming activities

Fig source: http://docs.oracle.com/javaee/7/tutorial/doc/jms-concepts003.htm
try (JMSContext context = connectionFactory.createContext();) {
    int count = 0;
    for (int i = 0; i < NUM_MSGS; i++) {
        message = "This is message " + (i + 1) + " from producer";
        TextMessage msg = context.createTextMessage();
        msg.setText(message);
        msg.setIntProperty("ID",count);
        if (((i+1) %2)==0) {
            msg.setStringProperty("msgType","EVEN");
        } else    msg.setStringProperty("msgType","ODD");
        context.createProducer()
            .setDeliveryMode(DeliveryMode.NON_PERSISTENT)
            .send(dest, msg);
        count += 1;
    }
}
System.out.println("Messages sent: " + count);
Some other JMS API features

- Control message acknowledgement
  - By JMS provider or by the client
- Message parameters
  - Persistent, priority, delay, and expiration
- Programming temporal destinations
- Nondurable versus durable subscription for subscribers
- Asynchronous sending

Generic question: how does the broker manage durable subscription?
Example of temporary queues for performance improvement

Common static queues for multiple clients

Separate static queues for multiple clients

Temporary queues

Outside the java world?

Source: http://docs.spring.io/spring-python/1.2.x/sphinx/html/jms.html
Recall

Would you use a JMS topic or queue?

Figure source: http://queue.acm.org/detail.cfm?id=1971597
Communication

ADVANCED MESSAGE QUEUING PROTOCOL
Overview

- MOM, but not language- or platform- specific
  - For Java, C#, Python, ….
  - Solving message interoperability in heterogeneous environments of MOMs
- Binary wire-level protocol for message exchange, rather than APIs
  - It does not include broker behaviors/capabilities but they were in the standard before version 1.0
- http://www.amqp.org
- Apache Qpid™
Core concepts – Message/Transport

- **Message representation**
  - Defined based on type systems for interoperability

- **Transport**
  - A network of nodes connected via links
  - Node: message storage, delivery, relay, etc.
  - Container: includes nodes

Figs source: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf
Core concept -- Transport

Figs source: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf
Example

- Get a free instance of RabbitMQ from cloudamqp.com
- Get code from: https://github.com/cloudamqp/java-amqp-example
- First run the test sender, then run the receiver

```
channel.queueDeclare(QUEUE_NAME, false, false, false, null);
for (int i = 0; i < 100; i++) {
    String message = "Hello distributed systems guys: "+i;
    channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
    System.out.println("[x] Sent " + message + ":'');
    new Thread().sleep(5000);
}
```

```
while (true) {
    QueueingConsumer.Delivery delivery = consumer.nextDelivery();
    String message = new String(delivery.getBody());
    System.out.println("[x] Received " + message + ":'');
}
```

Note: i modified the code a bit
Real code versus simulation

RabbitMQ Simulator
Use the drawing area below to represent your messaging topology. Drag messaging elements from the toolbox on the left to the canvas. To connect nodes, hold the ALT key (or SHIFT key) and drag from a source node to connect it to a destination node.

http://tryrabbitmq.com/
Performance

- "RabbitMQ Hits One Million Messages Per Second on Google Compute Engine"
  - [https://blog.pivotal.io/pivotal/products/rabbitmq-hits-one-million-messages-per-second-on-google-compute-engine](https://blog.pivotal.io/pivotal/products/rabbitmq-hits-one-million-messages-per-second-on-google-compute-engine)
  - [https://cloudplatform.googleblog.com/2014/06/rabbitmq-on-google-compute-engine.html](https://cloudplatform.googleblog.com/2014/06/rabbitmq-on-google-compute-engine.html)
- Using 32 nodes
- RabbitMQ is widely used in big industries!
http://mqtt.org

MESSAGE QUEUING
TELEMETRY TRANSPORT
(MQTT)
MQTT Overview

- OASIS Standard
- ISO/IEC 20922:2016 (Message Queuing Telemetry Transport (MQTT) v3.1.1)
- M2M Connectivity Protocol atop TCP/IP
- MQTT brokers enable publish/subscribe messaging systems
  - Publisher can publish a message within a topic that can be subscribed by many Subscribers

- Simple protocols
  - Suitable for constrained devices.
Protocol Features

- Lightweight protocol
  - Small message size
  - QoS
    - At most once, at least once and exactly once
    - Few commands/interactions: CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, DISCONNECT
      - Easy to implement

- Small footprint library
- Low bandwidth, high latency, data limits, and fragile connections
- Suitable for IoT (constrained devices/networks)

How QoS would impact the design of the subscriber?
Model and Implementation

- Different programming languages for OS/devices
  - Including Arduino, Nanode
- Mosquitto (http://projects.eclipse.org/projects/technology.mosquitto)
- Paho: http://www.eclipse.org/paho/
- RabbitMQ
- Apache ActiveMQ
- Cloud providers:
  - http://cloudmqtt.com (get a free account to learn MQTT)
Integration

MESSAGE ROUTING PATTERNS
Integration Issues

- We need several features implemented by MOM, consumer, or external systems

http://www.eaipatterns.com/
Example of supporting technology

Best practices for solving common problems: Integration Patterns

Also check: http://projects.spring.io/spring-integration/
Content-Based Message Routing: Camel/EIP

**Content-Based Router:** can be used to decide the right destination queue for a given message based on the message content.

**Dynamic Router:** can self-configure based on processing messages.

Source: https://camel.apache.org/content-based-router.html

Source: https://camel.apache.org/dynamic-router.html
Content-Based Message Routing: AMQP

Note: defined in AMQP 0-10
But not in AMQP 1.0

Some code example with RabbitMQ
Message Selector or Message Filter: filter unneeded messages

```java
TextMessage msg = context.createTextMessage();
msg.setText(message);
msg.setIntProperty("ID", count);
if ((count % 2)==0) {
    msg.setStringProperty("msgType","EVEN");
} else {
    msg.setStringProperty("msgType","ODD");
}

JMSCConsumer consumer = context.createConsumer(dest,"msgType =’EVEN’");
```

CAMEL/EIP: Message Filter

https://camel.apache.org/message-filter.html
Integration

TRANSFORMATION PATTERNS AND TOOLS
**Splitter and Aggregator**

**Splitter**: decompose a composite message into different messages

**Aggregator**: gather all correlated messages for a specific purpose then build a new composite message

Questions: for which scenarios/use cases we can use the above-mentioned patterns

[https://camel.apache.org/splitter.html](https://camel.apache.org/splitter.html)

[https://camel.apache.org/aggregator2.html](https://camel.apache.org/aggregator2.html)
How would you use splitter and aggregator with a set of microservices for a request?
**Envelope Wrapper and Normalizer**

**Envelope wrapper:** wrap a message before sending it into a messaging system and unwrap it after the wrapped message leaves the messaging system.

**Normalizer:** route all messages of a given type to a suitable Message Translator which transforms the message to the common format.

[Diagram of envelope wrapper and normalizer]

- [www.eaipatterns.com/EnvelopeWrapper.html](http://www.eaipatterns.com/EnvelopeWrapper.html)
- [camel.apache.org/normalizer.html](https://camel.apache.org/normalizer.html)
Content Enricher: obtain required/missing data then enrich the message with the newly obtained data

Content Filter: remove unimportant data items from a message or extract only needed information.

https://camel.apache.org/content-enricher.html

https://camel.apache.org/content-filter.html

Question: is it possible to send the to-be-enriched message to an external service to enrich it or to send the message to an external extraction service?
Logstash

- Codecs: stream filters within inputs or outputs that change data representation
- E.g.: multilines → a single event

Plug-ins

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Description</th>
<th>Logstash Plugin</th>
</tr>
</thead>
<tbody>
<tr>
<td>beats</td>
<td>Receives events from the Elastic Beats framework</td>
<td>logstash-input-beats</td>
</tr>
<tr>
<td>cloudwatch</td>
<td>Pulls events from the Amazon Web Services CloudWatch API</td>
<td>logstash-input-cloudwatch</td>
</tr>
<tr>
<td>couchdb_changes</td>
<td>Streams events from CouchDB's <code>.changes</code> URI</td>
<td>logstash-input-couchdb_changes</td>
</tr>
<tr>
<td>drupal_dblog</td>
<td>Retrieves watchdog log events from Drupal installations with DRB log enabled</td>
<td>logstash-input-drupal_dblog</td>
</tr>
<tr>
<td>elasticsearch</td>
<td>Reads query results from an Elasticsearch cluster</td>
<td>logstash-input-elasticsearch</td>
</tr>
<tr>
<td>eventlog</td>
<td>Pulls events from the Windows Event Log</td>
<td>logstash-input-eventlog</td>
</tr>
<tr>
<td>exec</td>
<td>Captures the output of a shell command as an event</td>
<td>logstash-input-exec</td>
</tr>
<tr>
<td>file</td>
<td>Streams events from files</td>
<td>logstash-input-file</td>
</tr>
<tr>
<td>ganglia</td>
<td>Reads Ganglia packets over UDP</td>
<td>logstash-input-ganglia</td>
</tr>
<tr>
<td>gelf</td>
<td>Reads GELF-formatted messages from GrayLog2 as events</td>
<td>logstash-input-gelf</td>
</tr>
<tr>
<td>gemfire</td>
<td>Pushes events to a GemFire region</td>
<td>logstash-input-gemfire</td>
</tr>
<tr>
<td>generator</td>
<td>Generates random log events for test purposes</td>
<td>logstash-input-generator</td>
</tr>
<tr>
<td>github</td>
<td>Reads events from a GitHub webhook</td>
<td>logstash-input-github</td>
</tr>
<tr>
<td>graphite</td>
<td>Reads metrics from the graphite tool</td>
<td>logstash-input-graphite</td>
</tr>
<tr>
<td>heartbeat</td>
<td>Generates heartbeat events for testing</td>
<td>logstash-input-heartbeat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Description</th>
<th>Logstash Plugin</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregate</td>
<td>Aggregates information from several events originating with a single task</td>
<td>logstash-filter-aggregate</td>
</tr>
<tr>
<td>alter</td>
<td>Performs general alterations to fields that the mutate filter does not handle</td>
<td>logstash-filter-alter</td>
</tr>
<tr>
<td>anonymize</td>
<td>Replaces field values with a consistent hash</td>
<td>logstash-filter-anonymize</td>
</tr>
<tr>
<td>cldr</td>
<td>Checks IP addresses against a list of network blocks</td>
<td>logstash-filter-cldr</td>
</tr>
<tr>
<td>cipher</td>
<td>Applies or removes a cipher to an event</td>
<td>logstash-filter-cipher</td>
</tr>
<tr>
<td>clone</td>
<td>Duplicates events</td>
<td>logstash-filter-clone</td>
</tr>
<tr>
<td>collate</td>
<td>Collates events by time or count</td>
<td>logstash-filter-collate</td>
</tr>
<tr>
<td>csv</td>
<td>Parses comma-separated value data into individual fields</td>
<td>logstash-filter-csv</td>
</tr>
<tr>
<td>date</td>
<td>Parses dates from fields to use as the Logstash timestamp for an event</td>
<td>logstash-filter-date</td>
</tr>
<tr>
<td>de_dot</td>
<td>Computationally expensive filter that removes dots from a field name</td>
<td>logstash-filter-de_dot</td>
</tr>
<tr>
<td>dissect</td>
<td>Extracts unstructured event data into fields using delimiters</td>
<td>logstash-filter-dissect</td>
</tr>
<tr>
<td>dns</td>
<td>Performs a standard or reverse DNS lookup</td>
<td>logstash-filter-dns</td>
</tr>
<tr>
<td>drop</td>
<td>Drops all events</td>
<td>logstash-filter-drop</td>
</tr>
<tr>
<td>elapsed</td>
<td>Calculates the elapsed time between a pair of events</td>
<td>logstash-filter-elapsed</td>
</tr>
<tr>
<td>cloudwatch</td>
<td>Aggregates and sends metric data to AWS CloudWatch</td>
<td>logstash-output-cloudwatch</td>
</tr>
<tr>
<td>csv</td>
<td>Writes events to disk in a delimited format</td>
<td>logstash-output-csv</td>
</tr>
<tr>
<td>datalog</td>
<td>Sends events to DataDogHQ based on Logstash events</td>
<td>logstash-output-datalog</td>
</tr>
<tr>
<td>datalog_metrics</td>
<td>Sends metrics to DataDogHQ based on Logstash events</td>
<td>logstash-output-datalog_metrics</td>
</tr>
<tr>
<td>elasticsearch</td>
<td>Stores logs in Elasticsearch</td>
<td>logstash-output-elasticsearch</td>
</tr>
<tr>
<td>email</td>
<td>Sends email to a specified address when output is received</td>
<td>logstash-output-email</td>
</tr>
<tr>
<td>exec</td>
<td>Runs a command for a matching event</td>
<td>logstash-output-exec</td>
</tr>
<tr>
<td>file</td>
<td>Writes events to files on disk</td>
<td>logstash-output-file</td>
</tr>
<tr>
<td>ganglia</td>
<td>Writes metrics to Ganglia's feed</td>
<td>logstash-output-ganglia</td>
</tr>
<tr>
<td>gelf</td>
<td>Generates GELF formatted output for GrayLog2</td>
<td>logstash-output-gelf</td>
</tr>
<tr>
<td>google_bigquery</td>
<td>Writes events to Google BigQuery</td>
<td>logstash-output-google_bigquery</td>
</tr>
<tr>
<td>google_cloud_storage</td>
<td>Sends events to Google Cloud Storage</td>
<td>logstash-output-google_cloud_storage</td>
</tr>
<tr>
<td>graphite</td>
<td>Writes metrics to Graphite</td>
<td>logstash-output-graphite</td>
</tr>
<tr>
<td>grafana</td>
<td>Sends metric data on Windows</td>
<td>logstash-output-grafana</td>
</tr>
<tr>
<td>hipchat</td>
<td>Writes events to HipChat</td>
<td>logstash-output-hipchat</td>
</tr>
<tr>
<td>http</td>
<td>Sends events to a generic HTTP or HTTPS server</td>
<td>logstash-output-http</td>
</tr>
</tbody>
</table>
Logstash Grok

Grok is for parsing unstructured log data text patterns into something that matches your logs.

Grok syntax:  %{SYNTAX:SEMANTIC}
Regular and custom patterns
A lot of exiting patterns:

https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns

Debug Tools: http://grokdebug.herokuapp.com/
Example with NETACT Log

29869;10/01/2017 00:57:56;;Major;PLMN-PLMN/BSC-401441/BCF-137/BTS-403;XYZ01N;ABC08;DEF081;BTS OPERATION DEGRADED;00 00 00 83 1111;Processing

Simple Grok

```
input { 
  file { 
    path => "~/tmp/alarmtest2.txt" 
    start_position => "beginning" 
  } 
} 
filter { 
  grok { 
    match => "{message} => "%(AlarmID)\%{DATESTAMP:Start}\%{DATESTAMP:End}\%{WORD:Severity}\%{NOTSPACE:NetworkType}\%{NOTSPACE:BSName}\%{NOTSPACE:StationName}\%{NOTSPACE:CellName}\%{NOTSPACE:AlarmInfo}\%{NOTSPACE:Extra}\%{NOTSPACE:AlarmStatus}" 
  } 
} 
output { 
  stdout { 
    csv { 
      fields => ["AlarmID","Start","Stop","Severity","NetworkType","BSName","StationName","CellName","AlarmInfo","Extra","AlarmStatus"] 
      path => "~/tmp/test-%{YYYY-MM-dd}.txt" 
    } 
  } 
} 
```
Apache Nifi

- From NSA
- http://nifi.apache.org/
- Main concepts:
  - Processor: components to handle data, such as download, store, transform, etc.
  - FlowFile: describes how different components are composed to create pipelines for data ingestion
  - Provenance (for data governance): see all usage records in detail
Apache Nifi

OS/Host

JVM

Web Server

Flow Controller

Processor 1

Extension N

FlowFile Repository

Content Repository

Provenance Repository

Local Storage

https://nifi.apache.org/docs.html
Example
Processing

COMPLEX EVENT PROCESSING
Dataflow programming and streaming processing

- Data exchange between tasks
  - Links in task graphs reflect data flows

- Streaming processing
  - Centralized or distributed (in terms of computing resources)
  - Various applications
  - CEP is just one type of applications of streaming processing

- Note: we will go further some advanced streaming processing in Lecture 5
Centralized versus distributed processing topology

Two views: streams of events or cloud of events

Centralized processing:
- Event cloud
- Usually only queries/patterns are written

Distributed processing:
- Code processing events and topologies need to be written
Goals of complex event processing

- Group and process events in a specific time (time) and space (size) constraints
  - Detect special situations
  - Finding correlation among events
  - Aggregation results
- Special case of streaming processing
WSO2 Carbon
CEP/Siddhi

**Performance**
Up to 6M Events/Sec on same JVM &
About 250K over Network

Source:
https://docs.wso2.com/display/CEP420
Common concept in these systems

- **The way to connect data streams and obtain events**
  - Focusing very much on connector concepts and well-defined event structures (e.g., can be described in XML, JSON, POJO)
  - Assume that existing systems push the data

- **The way to specify “analytics”**
  - Statements and how they are glued together to process flows of events
  - High-level, easy to use

- **The engine to process analytics requests**
  - Centralized in the view of the user → so the user does not have to program complex distributed applications
  - Underlying it might be complex (for scalability purposes)

- **The way to push results to external components**
If we
• specify a set of conditions for the window and events within the window
then we can
• get a set of events filtered from the window that match these conditions
Conditions: can be specified using an SQL-alike language or pre-defined functions
Event Representation, Streams and Views

- Event sources: via MOM, files, different IO adapters/connectors, etc.
- Event representation & views
  - POJO (Plain Old Java Object), Map, Object-array, XML
  - SQL-alike tables
- Event Stream
  - Events ordered based on their arrival times
- Event Sink
  - A component receiving events via its listener that declares some statements on interesting events
Windows and Times

Window size and slide

Batch/Tumbling Windows

## Flink CEP Patterns

<table>
<thead>
<tr>
<th>Pattern Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin(#name)</td>
<td>Defines a starting pattern:</td>
</tr>
<tr>
<td></td>
<td>Pattern&lt;Event, ?&gt; start = Pattern.&lt;Event&gt;begin(&quot;start&quot;);</td>
</tr>
<tr>
<td>begin(#pattern_sequence)</td>
<td>Defines a starting pattern:</td>
</tr>
<tr>
<td></td>
<td>Pattern&lt;Event, ?&gt; start = Pattern.&lt;Event&gt;begin(</td>
</tr>
<tr>
<td></td>
<td>Pattern.&lt;Event&gt;begin(&quot;start&quot;).where(...).followedBy(&quot;middle&quot;).where(...)</td>
</tr>
<tr>
<td></td>
<td>);</td>
</tr>
<tr>
<td>next(#name)</td>
<td>Appends a new pattern. A matching event has to directly succeed the previous</td>
</tr>
<tr>
<td></td>
<td>matching event (strict contiguity):</td>
</tr>
<tr>
<td></td>
<td>Pattern&lt;Event, ?&gt; next = start.next(&quot;middle&quot;);</td>
</tr>
<tr>
<td>next(#pattern_sequence)</td>
<td>Appends a new pattern. A sequence of matching events have to directly</td>
</tr>
<tr>
<td></td>
<td>succeed the previous matching event (strict contiguity):</td>
</tr>
<tr>
<td></td>
<td>Pattern&lt;Event, ?&gt; next = start.next(</td>
</tr>
<tr>
<td></td>
<td>Pattern.&lt;Event&gt;begin(&quot;start&quot;).where(...).followedBy(&quot;middle&quot;).where(...)</td>
</tr>
<tr>
<td></td>
<td>);</td>
</tr>
<tr>
<td>followedBy(#name)</td>
<td>Appends a new pattern. Other events can occur between a matching event and</td>
</tr>
<tr>
<td></td>
<td>the previous matching event (relaxed contiguity):</td>
</tr>
<tr>
<td></td>
<td>Pattern&lt;Event, ?&gt; followedBy = start.followedBy(&quot;middle&quot;);</td>
</tr>
<tr>
<td>followedBy(#pattern_sequence)</td>
<td>Appends a new pattern. Other events can occur between a sequence of</td>
</tr>
<tr>
<td></td>
<td>matching events and the previous matching event (relaxed contiguity):</td>
</tr>
<tr>
<td></td>
<td>Pattern&lt;Event, ?&gt; followedBy = start.followedBy(</td>
</tr>
<tr>
<td></td>
<td>Pattern.&lt;Event&gt;begin(&quot;start&quot;).where(...).followedBy(&quot;middle&quot;).where(...)</td>
</tr>
<tr>
<td></td>
<td>);</td>
</tr>
</tbody>
</table>

### Flink CEP Patterns

<table>
<thead>
<tr>
<th>Pattern Operation</th>
<th>Description</th>
</tr>
</thead>
</table>
| where(condition)  | Defines a condition for the current pattern. To match the pattern, an event must satisfy the condition. Multiple consecutive where() clauses lead to their conditions being ANDed:  
```java
pattern.where(new IterativeCondition<Event>() {  
    @Override  
    public boolean filter(Event value, Context ctx) throws Exception {  
        return ... // some condition  
    }  
});
``` |
| or(condition)      | Adds a new condition which is ORed with an existing one. An event can match the pattern only if it passes at least one of the conditions:  
```java
pattern.where(new IterativeCondition<Event>() {  
    @Override  
    public boolean filter(Event value, Context ctx) throws Exception {  
        return ... // some condition  
    }  
});.or(new IterativeCondition<Event>() {  
    @Override  
    public boolean filter(Event value, Context ctx) throws Exception {  
        return ... // alternative condition  
    }  
});
``` |
| until(condition)   | Specifies a stop condition for a looping pattern. Meaning if event matching the given condition occurs, no more events will be accepted into the pattern.  
Applicable only in conjunction with oneOrMore()  
**NOTE:** It allows for cleaning stale for corresponding pattern on event-based condition.  
```java
pattern.oneOrMore().until(new IterativeCondition<Event>() {  
    @Override  
    public boolean filter(Event value, Context ctx) throws Exception {  
        return ... // alternative condition  
    }  
});
``` |

Data

<table>
<thead>
<tr>
<th>station_id</th>
<th>datapoint_id</th>
<th>alarm_id</th>
<th>event_time</th>
<th>value</th>
<th>valueThreshold</th>
<th>isActive</th>
<th>storedtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1161115016</td>
<td>121</td>
<td>308</td>
<td>2017-02-18 18:28:05 UTC</td>
<td>240</td>
<td>240</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>1161114050</td>
<td>143</td>
<td>312</td>
<td>2017-02-18 18:56:20 UTC</td>
<td>28</td>
<td>28</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>1161115040</td>
<td>141</td>
<td>312</td>
<td>2017-02-18 18:22:03 UTC</td>
<td>56</td>
<td>56</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>1161114008</td>
<td>121</td>
<td>308</td>
<td>2017-02-18 18:34:09 UTC</td>
<td>240</td>
<td>240</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>1161115040</td>
<td>141</td>
<td>312</td>
<td>2017-02-18 18:20:49 UTC</td>
<td>56</td>
<td>56</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>
Simple example

```java
final RMQConnectionConfig connectionConfig = new RMQConnectionConfig.Builder()
    .setUrl(env[0]).build();
final DataStream<String> stream = env
    .addSource(new RMQSource<String>()
        .connectionConfig, env[1], false,
        new SimpleStringSchema())
    .setParallelism(1);

DataStream<AlarmEvent> btsStream;
btsStream = stream.flatMap(new BTParser());

Pattern<AlarmEvent, ?> pattern = Pattern.<AlarmEvent>begin("start")
    .where(new SimpleCondition<AlarmEvent>()
        @Override
        public boolean filter(AlarmEvent value) throws Exception {
            System.out.println("Start event");
            return value.alarm_id.equals("308");
        }
    ).next("middle")
    .followedBy("end")
    .where(new SimpleCondition<AlarmEvent>()
        @Override
        public boolean filter(AlarmEvent value) throws Exception {
            System.out.println("End event");
            return value.alarm_id.equals("303");
        }
    )
    .within(Time.seconds(300));

PatternStream<AlarmEvent> patternStream;
patternStream = CEP.pattern(btsStream.keyBy(new AlarmKeySelector()), pattern);

final DataStream<String> alerts = patternStream.select(new PatternSelectFunction<AlarmEvent, String>()
    @Override
    public String select(Map<String, List<AlarmEvent>> pattern) {
        AlarmEvent first = pattern.get("start").get(0);
        AlarmEvent second = pattern.get("end").get(0);
        final String result = "Detected: " + first.toJSONString() + " --> " + second.toJSONString();
        System.out.println("FOUND: " + result);
        return result;
    }
);

RMQSink<String> sink = new RMQSink<String>(
    connectionConfig, env[2],
    new SimpleStringSchema());
alerts.addSink(sink);
```
Monitoring

![Apache Flink Dashboard](image)

Source: Custom Source -> FI at Map
Parallelism: 1

FlatSelectCepOperator -> Sink: Unnamed
Parallelism: 1

<table>
<thead>
<tr>
<th>Subtasks</th>
<th>Task Metrics</th>
<th>Watermarks</th>
<th>Accumulators</th>
<th>Checkpoints</th>
<th>Back Pressure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Start Time</th>
<th>End Time</th>
<th>Duration</th>
<th>Name</th>
<th>Bytes received</th>
<th>Records received</th>
<th>Bytes sent</th>
<th>Records sent</th>
<th>Parallelism</th>
<th>Tasks</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-03-20, 21:06:44</td>
<td>2018-03-20, 21:09:13</td>
<td>2m 28s</td>
<td>Source: Custom Source -&gt; Flat Map</td>
<td>0 B</td>
<td>0</td>
<td>1.90</td>
<td>8</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2018-03-20, 21:06:44</td>
<td>2018-03-20, 21:09:13</td>
<td>2m 28s</td>
<td>FlatSelectCepOperator -&gt; Sink: Unnamed</td>
<td>1.90 KB</td>
<td>8</td>
<td>0 B</td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Results

DST 2018
SQL-alike CEP

- We can register/view stream as a table (like SQL)

- Then apply SQL-alike statements with windows for detecting events and patterns

- Tools: Esper, WSO2, and certain streaming databases
Example of WSO2 Siddhi

Pass-through

```
from <stream-name>
select ({attribute-name}] "*)
insert into <stream-name>
```

Filters

```
from <stream-name> {<conditions>}
select ({attribute-name}] "*)
insert into <stream-name>
```

Windows

```
from <stream-name> {<conditions>}
    #window.<window-name>({<parameters>})
select ({attribute-name}] "*
insert [<output-type>] into <stream-name>
```

Source: https://docs.wso2.com/display/CEP420/SiddhiQL+Guide+3.1
SQL-alike conditions

@Import('mobifonetraingingopensignal:1.0.0')

declare stream inStream (meta_USERPHONE int, meta_TIME long, correlation_lat float,
correlation_lon float, GSM_BIT_ERROR_RATE float, GSM_SIGNAL_STRENGTH float,
LOC_ACCURACY float, LOC_SPEED float);

@Export('OutputSignal:1.0.0')

declare stream OutputSignal (avgSignalStrength double, avgBitRateError double);

from inStream#window.lengthBatch(5)
select avg(GSM_SIGNAL_STRENGTH) as avgSignalStrength, avg(GSM_BIT_ERROR_RATE) as avgBitRateError
insert into OutputSignal;
A data pipeline of stream receivers $\rightarrow$ event processor $\rightarrow$ event publishers
Example with WSO2 Carbon CEP

Source → Service → Sink

Language + UI specifications
Get a high-level view

Check: http://de.slideshare.net/alessandro_margara/processing-flows-of-information-debs-2011
Datalake with messaging

Figure source: Data Lake for Enterprises by Pankaj Misra; Tomcy John Published by Packt Publishing, 2017
Cloud services and big data analytics

Data sources (sensors, files, database, queues, log services) → Messaging systems (e.g., Kafka, AMQP, MQTT) → Storage and Database (S3, InfluxDB, HDFS, Cassandra, MongoDB, Elastic Search etc.) → Stream processing systems (e.g. Apex, Storm, Flink, WSO2, Google Dataflow) → Batch data processing systems (e.g., Hadoop, Airflow, Spark) → Elastic Cloud Infrastructures (VMs, dockers, OpenStack elastic resource management tools, storage) → Operation/Management/ Business Services → Warehouse Analytics
Data Processing Framework

- Batch processing
  - Mapreduce/Hadoop
  - Scientific workflows
- (Near) realtime streaming processing
  - Flink, Apex, Kafka SQL, Storm
- Hybrid data processing
  - Summingbird, Apache Kylin
  - Impala, Storm-YARN
  - Apache Spark

Take a short read: http://www.infoq.com/articles/stream-processing-hadoop
Conceptual View

Analytics (Application Level)

Data Processing Frameworks

- Streaming/Online Data Processing
- Batch Data Processing
- Hybrid Data Processing

Analytics, Tools, Processes & Models

Decision → Data Analysis

(Near) realtime data → Static data
Recap

So how can you use messaging techniques for complex distributed applications/systems?

- Reactive patterns
- Asynchronous communications
- Large-scale integration
- Big data
- ?
Further materials

- [http://kafka.apache.org](http://kafka.apache.org)
- [http://www.eaipatterns.com](http://www.eaipatterns.com)
- [http://docs.oracle.com/javaee/7/tutorial/doc/home.htm](http://docs.oracle.com/javaee/7/tutorial/doc/home.htm)
- [http://docs.oracle.com/cd/E13157_01/wlevs/docs30/epl_guide/index.html](http://docs.oracle.com/cd/E13157_01/wlevs/docs30/epl_guide/index.html)
Thanks for your attention

Hong-Linh Truong
Faculty of Informatics, TU Wien
hong-linh.truong@tuwien.ac.at
http://www.infosys.tuwien.ac.at/staff/truong