Meadowview: A Middleware for Event Clouds

cjeongkr@gmail.com

November 9, 2013

Event Cloud

An **event cloud** is an infrastructure for distributed computing where computational entities can

- register new events[1]
- inject events[2]
- inject event rules
- subscribe to events

[1] Should we force registration of event before pub/sub?, Can we make event instances to self-contained (incl. data layout) and use dumb string-matching-based event matching?

[2] more precisely, event instances

Event Cloud (Cont.)

Components of Event Cloud

There are a few components which consitute a event cloud.

- An **event** is a class of status change that are of interest. A particular occurrence of an event is called its **event instance**[3].
- An **event rule** specifies a predicate over events and a finite set of events which will be generated as the evaluated result of the predicate.
- An **endpoint** is a computational entity which can produce or consume events.

Analogy to Programming Languages

A programming language consists of variables (events), which contains values (event instances). Event rules define the intended semantics of the event cloud, just like a program has its own meaning. Interpreter executes the program (event rules).

[3] Sometimes, we will use the word "event" instead of "event instances" in case the intended meaning is obvious from the context.

Event Rules

Event Rules

An **event rule** can be specified as follows:

```
process EventRule(Event e) {
  if (e.value > 4.0)
    -> LargeValueEvent(e.value);
  else
    -> SmallValueEvent(e.value);
}
```

In the event rule, only simple arithmetic and comparison operators are allowed. No complex computations, including function calls, are not allowed. However, this would not harm expressiveness of the event rule since we can always let endpoints do complex computations.

End-to-end principle

The decision to exclude any complex computations in the event rule is to make the event cloud responsible for handling only very simple event rules. If, for any complex event predicate or processing is needed, we can always create a new event which will be routed to endpoint (event consumer) which will perform the computation and then inject the result as another event.

Event Graphs

An **event graph** is defined as $G = (V, E, R)$.

- A vertex $v \in V$ is either an event, a source, or a sink.
  - Source vertices are those with indegree 0. A source is an endpoint in the event cloud, which generates event instances.
  - Sink vertices are those with outdegree 0. A source is an endpoint in the event cloud, which consume event instances.
  - No vertex can both generate and consume event instances. Therefore, any cycle in the graph does not involve any endpoint.
- An edge $(u, v) \in E$ means that any instance of event u injected into event cloud may generate an event instance of v.
- $R : E \to P$ is a mapping from edges to event predicates. This is used to determine whether or not to fire the edge.

Event Engine

An **event engine** is responsible for execution of event graphs, which represents dependencies between events and firing semsntics of the event cloud.
That is, an event engine is an *interpreter* of *event graphs*.

Event Graphs vs Data Flow Graphs

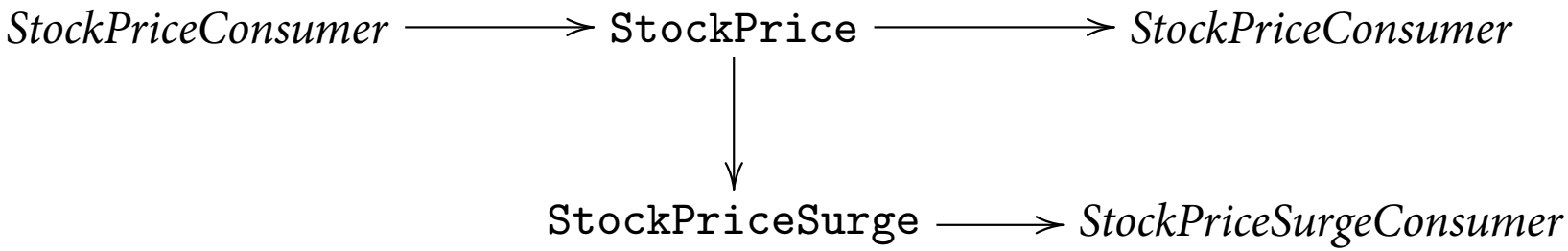
Event graphs are very different from data flow graphs. In data flow graphs, a vertex is an operator which performs computation and data (events) flow along the edges. That is, *data flow graph itself is an engine*. However, In event graph, *event graph is a target code to be executed by an engine*.

Simple PubSub Example

Suppose there is only one event, `StockPrice`, with one producer and one consumer. In this case, the event graph consists of a single vertex.

Another PubSub Example

Now there are two events, `StockPrice` and `StockPriceSurge` with one producer and two consumers.



`StockPriceSurge` is an example of an event which is generated not by a producer but by an event rule such as below:

```
process StockPriceWatch(StockPrice s) {
  if (s.price > 4.0)
    -> StockPriceSurge(s.symbol, s.price)
}
```

Usages of Event Cloud

Event detector/reactor

Simple event tap program which attaches a UNIX program to event cloud.

```
event_gen <program> "finished"
event_detect <program> "finished"
```

Event-Driven Distributed Programming

Adoption of event cloud greatly simplifies the deployment of distributed applications. For example, consider a farm which consists of n machines. When m jobs is to be distributed across these machines, all all necessary handshaking can be implemented with event deliveries using Meadoew library.

Workflow system

The event cloud can be used as an infrastructure for workflow systems. Any front-end GUI workflow tools can generate events and event rules and inject them to the event cloud.

Usages of Event Cloud (Cont.)

Event Auditing through Logs

The event cloud itself does not store event instances. As soon as event instances are created, when all its consumers are notified of these instances (including the transfer of payload of event instances), they are destroyed. However, one can easily create a new endpoint which listens to events of interest and stores the events in a persistent storage system.

Internet of Things

When the Internet of Things becomes a reality, devices will talk to each other. For this, we need a language.

Complex Event Processing

Complex event processing systems generate complex events based on a finite set of past occurrences of events. These events are "value-added events" which contains more useful information than mere events.
Then, HOW? We want the event engine to be STATELESS!