Event Deriver

You are building Event Driven Systems. You have a set of related events that you can tie together into a logical set (perhaps through application of the Event Joiner Pattern) that you are interested in.

When you have a set of related events, there is often a need to respond based not only on the occurrence of a single event, but on the occurrence of several events in a predefined sequence.

For instance, consider a system whose responsibility it is to detect a condition in a server that may indicate that the server is overheating. Let's say the server contains a temperature sensor. Now, a single temperature measurement (a temperature event or an "over limit" event) may not be interesting. Someone may have just momentarily put their hot cup of coffee on top of the server for a moment and then lifted it off. Or someone may have blocked the air flow into a server temporarily by moving a piece of equipment past a vent.

However, if you have several "over limit" temperature events in a row, over a period of time, then you may have a condition that is detrimental to the functioning of the server. But you can only determine that if you can somehow link each individual temperature event to other events of that type.

How do you respond to a series of related events by generating a new event that depends on the occurrence of many different events over time?

There are several things that complicate this:

- You need to be able to define the sequence of events you are interested in both in terms of the actual events and their properties and in terms of the set of events that is of interest.
- You need to be able to avoid "false positives" that can be caused by too simple rules. For instance, a simple counter of events of a type would not suffice since the relationship between the events may require temporal comparison (for instance, it needs to be no more than 30 seconds since the last event).

Therefore,

Create a process, e.g., an event deriver, that is responsible for listening to events, determining if a particular sequence of events has taken place, and creating a new, derived event as a result.

There are at two different, but closely linked approaches to solving this problem that arrive at the same end. The first is the state-machine approach. In that approach, a state machine instance is created upon receipt of the first event matching a particular correlation id. As new events are received for that same correlation id, the state machine moves along its lifecycle. At various points in the lifecycle of the state machine, new events (Derived Events) can be generated as a result of a state transition, reflecting that a number of linked events have been processed in a particular sequence. As each state is matched, the state itself records information about the conditions that led to its transition (information about the last event).

A second approach that has the same effect is to have a rules engine that evaluates new events that enter the deriver – as new rules fire, they can either produce intermediate results that may affect the result of later rules, or they may produce derived events as a result. In this case, the intermediate results need to include all the necessary information to be able to allow later rules to fire – this may need to include information about the last events received.

A "canonical" use of the Event Deriver Pattern is in detecting financial fraud of various types. For instance, consider the following problem. When a person's credit card or ATM card is stolen, then the thief may try to use it as often as possible within a short period of time before it is reported as being stolen. Thus, a large number of cash withdrawals from an ATM card in a short period of time may indicate fraud. Likewise, if a card is stolen and the number posted on the internet, then it may be used from many different locations at once. This would also be an indicator of fraud.

You could solve this problem by applying the Event Deriver pattern. An Event Deriver would watch the stream of events looking for particular sets of attributes in the events. For instance, as it sees two ATM withdrawals for the same account either in a short time frame, or widely separated in distance, it may increment a potential fraud count. When the potential fraud count reaches a limit, it generates a new "stolen card" event and places that on the backbone.

An Event Deriver becomes a new source of events that are different than the ones to which it responds. As such, Event Derivers may bridge between different event channels or even different Event Backbones. The Event Aggregator pattern discusses how an Event Deriver is part of an architecture of bridging Event Backbones.