Event Joiner

You are building applications that will follow an Event Driven Architecture.

Events are tricky and often transient things. There are many cases where a single event is only interesting when considered in the context of other events that may have previously occurred or may occur later. Connecting together events where unique identifiers are not consistent (sometimes called event correlation) is handled in many every day processes.

Consider a mortgage application process which is monitored for successful completion. As the application is reviewed by different individuals the mortgage application number is typically used as the unique ID. As background checks are done by a 3rd party, a different unique ID may be used to reference the customer. When the 3rd party returns their findings, this information needs to be correlated with other events which are already being correlated together. The unique ID used by the bank and the 3rd party may not be known until the time the 3rd party is engaged to investigate the individual. It may actually be dynamically determined based on some unique ID generator or it may be a different piece of information from the mortgage application.

Another example would be in the healthcare industry. When a patient comes in for surgery or some type of overnight stay, the hospital needs to track all types of actions which occur around the patient in order to correctly and fully bill the patient. As the patient receives treatment, medicine and meals, all of these activities can potentially be generate events through RFIDs. Many of these events may not have the specific patient ID associated with them as they are specific to the medicine cart or meal cart. These events must be correlated based on different IDs to track that the proper care is being provided and the proper billing is recorded.

Finally, consider the basic problem of connecting log events across different products like a web server, an application server and a database server so that you can view a single customer “transaction” across all three. The issue faced here is the same as the issue faced previously.

How do we connect events separated by time, space, or other distances to other events in order to determine if the combination of events is interesting?

With events coming at different times, from different sources and different technologies and over different protocols, the ability to identify an individual event which is to be considered in the context of other events is a challenge To identify the event with the context, a relationship needs to exist between the incoming event and the context and then recognized when the event is processed. Relationships are typically defined in the format of 0-1, 1-1, 0-n, or n-n with all formats requiring a unique identifier on all relationship objects. For events and event contexts, the relationships are limited to 1-1 and 1-n with 1 event to 1 event context or 1 event to n event contexts. For both types of relationships, a unique identifier must be associated with the event and a unique identifier must be associated with the event context(s).

The unique identifier for the event could be based on a number of things.

1. Event payload which is typically based on application data or specific to the application emitting the event.
2. Event metadata which is specific to the event emitter instance
3. Combination of 1 and 2

The unique identifier for the event context can be set by a number of things.

1. Internally generated unique identifier
2. Incoming event payload
3. Incoming event metadata
4. Combination of 2 and 3

Each event context must be started with at least one of the above unique identifiers. As additional events are received they will be checked for unique identifiers which match the relationship instance for an active event context. If the event does not have a matching unique identifier, the event will not be associated with the event context.

For many large, distributed solutions which are focused on processing events from a variety of technologies and event emitters, it becomes more difficult to specify a unique identifier in the event which matches the relationship instance for an active event context. As a business transaction flows through the different distributed technologies, session ID or transaction IDs or other metadata is not

readily passed. Different protocols do not always have the capacity to pass and/or receive metadata in header packets. Also the way in which information is passed in headers is not defined in a standard way. Also as the business transaction is executed, the business data which was used in the event payload at one point of the application may not be available at a subsequent point. Without the same business data available at that point, a unique identifier which was associated with the relationship instance will not be available and will not be included in an emitted event.

Correlating an event of this type will not be possible unless the appropriate event context has an additional unique identifier that can be matched. The additional unique identifier however, may not be known when the event context is started or within the context, it may not be appropriate to have an additional unique identifier recognized until a specific time. When an additional unique identifier is needed, there are a number of options for specifying the value.

Specified in a previously received event and set as an additional of unique identifier

Calculated from a previous event payload or metadata

Received through a response from a request sent to a service provided a unique identifier.

Therefore,

Build event joiners whose job it is to tie together different events that are part of a united stream. Since each event type may differ in its information content, there should be a different event joiner for each event type.

All Event joiners performs the same general sequence of actions:

1. Generate a common correlation identifier (Hohpe, 163) from the information in that particular event type.
2. Append the correlation identifier to the event.
3. Either publish the event (which now includes the generated correlation identifier) onto a new event topic, or place the new event in a persistent event store.

Event joiners operate on the “edges” of different event streams, as events move from one type to another. For instance, consider the following problem. If we are trying to correlate log events from an HTTP Server, an Application Server, and a Database Server, then we would need an Event Joiner that begins by looking at the events coming from the HTTP Server. From the information in the HTTP Request a unique correlation identifier will be generated for each unique request. When the log messages are generated in the Application Server, then the joiner working on the Application Server events can use the information in the HTTP request sent to the Application Server to identify which internal thread id corresponds to the correlation id from the HTTP Server and can use this internal linkage to append the common correlation id to the WAS log events. Finally, since the Database Server joiner can use the information about the database request (userid and SQL, at least) to correlate the log events in its event stream to those from the Application server and the HTTP Server.

Once the events have been added to the second event conduit or stored in the persistent event store then other processes can act on that stream, for instance an Event Deriver can react to a series of related events.

This pattern can be used for bringing seemingly unrelated events together into an event context in which they can be correlated in order to generate additional events.