Event Driven Architecture

To put it at a very basic level, event can be defined as a basic change in state. Let us take the example of a resort booking service. Whenever the customer, books a room, the state changes from “available” to “occupied”. In this case it’s not just the room state that needs to be updated, but other services too need to be notified of the change, like say Room Availability, Billing, Service Department. And this is handled by the system architecture, which is what sums up the event driven architecture paradigm, that handles the production, consumption, detection of and reaction to events.

We need to understand that events just occur, they do not travel, and the event driven architecture here, involves the design and implementation of applications that transmit events among loosely coupled services and software components. At a very high level, the event driven system consists of event emitters(agents) which produce the event, event consumers( sinks) and event channels. The emitter's job is to detect and transmit the events, it has no idea who the consumers are, in fact it does not even know what happens to the event. If we take our Resort Booking System, the Room Booking Service is the Emitter that transmits a request to book the room based on customer input. Now it does not even know if the rooms are available or not, or the charges of the room, or the billing. That job is to be done by the sinks or Event Consumers, as soon as the event is presented. Now the consumer could either generate its own reaction to the event, or filter and pass it to another service. In this case the Event Consumers would be say a Room Availability Service, that responds to the event by sending status of Available or Occupied. Based on the status, the Billing Service is invoked that calculates the charges and sends it to the Room Booking Service. Assuming the customer pays to book the Room, the event is passed to another consumer Payment Service, which in turn might invoke a Gateway Service, depending on customer mode of Payment. Once payment is done and Room is booked, the Notification Service notifies customer of Room Booked. In the event of Room not being available, a Notification could be sent of the same.

Now this transmission of the events between Emitter and Consumer is done by the Event Channel, that could either be message-oriented middleware or point to point communication. The advantage with this approach is that horizontal scalability is simplified, as the application state can be copied across multiple snapshots. For eg if we take our Resort Booking application, typically the booking sees a spike during holiday season or say long weekends. Instead of adding more nodes to handle the rush, we can took a copy of the application state, run it with a stream of events.

Events are made up of two parts, event header that has information like event name, event time, type of event, and the event body that produces details of state change detected. Again, the event body does not contain the business logic. For eg the Room Availability Service will only notify of change in state say from Available to Occupied.

Now when we look at the different layers of the event driven architecture, it broadly has four logical layers, beginning with the sensing of event, to creation of an event structure and ending with a set of reactions to the event.

**Event Producer**

The first logical layer, which senses an event and represents as a message. It could be say an email client, a monitoring agent. In our case the Room Booking Service would be the producer, that detects any requests made by user for room and converts into a standardized message. The requests could be made via email, mobile app, online portal.

**Event Channel**

The second logical layer, basically a mechanism to propagate information from an event producer to consumer or sink. It could be an email, XML format, API request, as it .queues the events.

**Event Processing Engine**

It is the logical layer that identifies the event and then channels to the appropriate handler. For eg when Room Booking Service sends a request, it identifies that this has to be channeled to Room Availability Service first.

**Downstream Event Driven Activity**

The logical layer where the event result is shown, for eg, when customer books a room during peak season, warning could be sent of low availability. However this is more an add on and not necessarily needed.

 When it comes to event processing, there are three styles basically.

**Simple Event Processing**

Basically, events related to direct specific, measurable changes of condition and used to drive the real time flow of work.

For eg in our application, Customer books a room, and event is triggered based on availability.

**Complex Event Processing**

This allows patterns of simple and ordinary events to be considered to infer a complex event, has occurred. The event correlation may be casual or temporal, and requires use of sophisticated event interpreters, event pattern definition.

For eg the Resort Booking System, would have to take into account the sudden spike in demand say during Festival seasons or Long weekends, or a mass booking like for a company event or a wedding function.

**Event Stream Processing**

This is typically a mix of Simple and Complex event processing, where both ordinary and notable events happen. Ordinary events( for eg orders), are screened and streamed to subscribers and used to drive the real time flow of information.

Basically the advantage of event driven architecture is it’s loose coupling and wide distribution, primarily due to the fact that an event can exist anywhere, the event itself does not know the consequences. It ha s a loose coupling within space, time and synchronization, providing a scalable infrastructure for information exchange and distributed workflow,