CHAPTER 2
EVENT META-MODEL AND EVENT TEMPLATE

2.1 INTRODUCTION
This chapter explains the proposed Event-based modeling framework and its objectives, in comparison to other frameworks and methodologies for OOA presented in Chapter 1. It introduces the concept of events, their types and significance as described in [57, 128, 133, 135, 138, 165, 166]. The proposed Event Meta-model, its principle and role in the Event-based framework is also explained. The detailed structure of proposed Event template is described which is used for documenting events. A critical comparison of an Event template with a Use Case template [151] is also explained.

2.2 EVENT-BASED FRAMEWORK
According to [171], a framework is a meta-level (a higher level of abstraction) concept through which a range of other concepts, models, techniques, methodologies can either be clarified and/or integrated. Framework is something which consists of guidelines for supporting the various phases of software development process, various templates and tools for capturing requirements, modeling, designing, generating test cases etc. On the similar lines this section explains the Event-based framework for OOA.

The primary foundation of the proposed Event-based framework is events occurring in a system. Events have been chosen as a starting point due to several reasons

(a) It is more realistic to use events than processes for requirement analysis and conceptual modeling.

(b) Events help the analysts or OO design team to determine which events should be allocated to operations on data centric persistent classes. This is so because events act upon many classes and conversely, the same class may be acted upon by a variety of seemingly unrelated events.
(c) Events use a technology-independent stimulus-response modeling technique, while deferring interaction design. For example, the event “Customer requests shipment status” can occur using a variety of technological media. Customers can request their shipments status by calling a customer service representative, requesting status via an Internet website, punching their way through the Interactive Voice Response (IVR) system, using Short Message Service (SMS) messaging, or can even send an Electronic Data Interchange (EDI) transaction but the event’s essential business policy remains unchanged. This helps to capture the essence of the business policy in the early stage of analysis and defer the design of dialog or interaction until the point where the technology is declared.

(d) Analysis of unexpected or expected events often uncovers important policy earlier in a project. For instance, what is to be done on the failure of expected or time-triggered events? Thus, event modeling lets a user create analysis specification that has more value to business in a long run. Also work in [138] states that events are object history. Event occurrences are of a particular kind or type called an event. Therefore, by recording events, one can keep the state-change history of any object and even use this history for the restart/recovery of systems. Events, thus, provide us with an object life accounting.

2.3 EVENT DEFINITION, TYPES AND THEIR SIGNIFICANCE

This section introduces the concept of events, their types and significance as described in [57, 128, 133, 135, 138, 165, 166]. Significance of an event is explained in terms of what operations are triggered on the participating classes by that event. The concept of event itself has been widely used to model software. Our approach is adopted from Stephen M. McMenamin and John F Palmer’s “Event Partitioning” method that uses events as starting point to create Data Flow Diagrams (DFDs) during structured analysis of a system [135].

According to event-partitioning approach, an event is a “stimulus” that occurs in an outside world and to which a system must respond. Each event is labeled with a F, a T, or a C to show whether an event is a flow-oriented event, a temporal event, or a control event respectively. A flow-oriented event is one which is associated with a data flow; that is, the
system becomes aware that an event has occurred when a piece of data (or possibly several pieces of data) has arrived. For example, Customer places order and Customer sends payment. In these cases, arrival of an order or a payment in a system indicates that flow-oriented events have occurred. Temporal events are triggered by the arrival of a point in time. For example, Customer needs (monthly) statement. Accounting needs (daily) cash receipts. Control events are events that occur when something happens inside a system and the system must initiate some process in response for this event. For example, Book reprint order arrives at warehouse as result of drop in number of copies of a book.

According to [138], events can be internal, external or temporal. An internal event occurs as a result of an operation that is within the analyst’s domain. For example, Book transferred event occurs as a result of a Transfer Book operation. An external event is the result of an operation that is external to the analyst’s domain. External events are identified when they have an impact on the analyst’s system. For example, an event request to check out library books is an external event for the library but it affects the count of books on loan. Temporal events are results of clock operations. Clock operations are operations that emit a specified pattern of clock-tick events. Since the actual clock operation can be internal and external to the analyst’s domain, temporal events can be represented either as external-temporal or as internal-temporal. Figure 2.1 shows the concept of events depicted in [138].

![Figure 2.1: Expressing internal, external, and temporal events [138.](image)

As per work done in [128], events are stated in subject-verb-object pattern. Some actor does something to something, e.g., “Customer places order,” “Sales Manager denies credit
request,” “Marketing Department changes prices.” In this thesis, we have considered this pattern to identify events from requirements. The work states that an event must pass following five tests before it is accepted onto the event list for a project:

1. An event occurs - at a specific moment in time.
2. An event occurs in the environment, not inside the system
3. The event is under the control of the environment, not the system
4. The system can detect that the event occurred
5. An event is relevant when a system is chartered to do something about it.

According to [128], the analyst can also classify the event as “unexpected” or “expected.” In case of unexpected events, a business (or a system) never knows when a particular instance of an event will occur. “Customer places order” is a classic unexpected event in most industries. Expected events on the other hand, are results of some predecessor events having established a window of expectation in a system during which a particular instance of an event is anticipated to occur. For example, “Warehouse ships order” is expected to occur based on the previous event of “Customer places order” having informed the warehouse to ship. Expected events are interesting because they help to define important business policy resulting from their failure. Temporal events are always expected events because they are result of passage of time exceeding a schedule established in a system by predecessor events.

Perspective defined in [133, 172, 173] states that an event is an occurrence at specific time and place and should be remembered by the system. An event is a set of activities that are performed either fully (when the pre-conditions are satisfied) or not at all (when one or more of the pre-conditions is/are not satisfied).

[174] states that an event is invoked by a stimulus which can be an actor or by a point in time reached in a system. [175] states that an event has an effect on the state of a system. The effect may be in the form of creating or deleting objects, or changing the state of
existing objects. An event is a record of system activities with attributes, significance and relativity [161].

According to [133] events can be external, temporal and system. External event occurs outside the system, usually initiated by the external agent (person or organizational unit or system user). For example, ‘Customer places an order’, ‘Customer wants to check the item availability’, ‘Merchandising creates a new catalogue’ etc. Here, customer is an external agent and as a result of this event, a new order is generated in a system. Temporal events are generated automatically by a system on reaching a given point of time. They do not occur on fixed date. Time of occurrence could also be relative to some other events occurrence. There is no need for external agent to trigger temporal events. Temporal events include internal or external outputs needed from time to time. For example, ‘System produces biweekly payroll’, ‘Time to produce order summary reports’, ‘Time to produce customer concession reports’ etc. System events occur when something critical happens inside a system that triggers the need for processing. These events monitor system in order to detect or respond to external systems, devices or another objects. System events are consequences of external events. For example, Order event in the above example, reduces stock in an inventory that results in generating a state event ‘Reorder point reached for that product’. Time cannot be predicted for System events. Most of the events in a general application domain are external and temporal. System events are more common in the domain of real time systems. For example in process control system, if vat of chemical is full, then a system event, ‘turn off the fill valve’ is generated.

From the different perspectives on events, it is clear that Events trigger all the processing in a system, so identifying and analyzing events is a good starting point in OOA of requirements. Events can affect state of objects (attribute or number of instances), relationships among objects or both. Our Event-based methodology uses a perspective that considers events to be External events, temporal events and System events.
2.4 PROPOSED EVENT-META-MODEL

A Model is an abstraction of a phenomenon in the real world; a Meta-model is yet another abstraction, highlighting properties of a model itself. A Meta-model is a precise definition of the constructs and rules needed for creating semantic models. Meta-models are always made for a particular purpose. A Model conforms to its Meta-model in a way a computer program conforms to the grammar of a programming language in which it is written. Meta-modeling in software engineering is the analysis, construction and development of the frames, rules, constraints, models and theories applicable and useful for modeling a predefined class of problems. In the past, several Meta-models are developed for different purposes. The next sub section describes some of those approaches.

2.4.1 Meta-Modeling Approaches

In [176] Meta-model is used in object-oriented modeling method for workflow applications. Authors have used Event Meta-model to illustrate and realize both control and distribution mechanism in the logical modeling of workflow applications. In [177] Meta-model of UML is presented that describes the constituents of all well-formed models that may be represented in UML, using UML itself. In [178] authors have used meta-model concept to integrate eight software system developments to represent strategic level development of a software system for Project Managers to operate on projects. In [179] a meta-modeling approach is used to explore the application of model integrated computing in developing web services. In [180] a Meta-model of a controlled natural language for interactive systems is presented for requirements specifications. It is called Project IT-Requirements Specification Language (RSL). Authors in [181] have developed a rule based business activity Meta-model to capture, store and manage business knowledge in the form of business rules in order to trace them from their origin in business environment to their implementation in IS. A requirements Meta-model is developed in [182] with a focus on stakeholder requirements interaction. In [183], a requirements description Meta-model called Requirement Description (RD)-Meta-model is proposed that integrates an Activity graph Meta-model and a Use case Meta-model to free engineers from the routine work of defining similar Use cases repeatedly and at the same time keeping consistency in
requirements elicitation. In [184], a Meta-model is proposed to support automatic
generation of conceptual database design tool software for a variety of data models. In
[185], authors have used Meta-model for requirement statements for embedded systems to
ensure applicability of natural language patterns in reducing ambiguity.
Unlike the objective of all the approaches that have used meta-model, our event-based
framework has proposed an Event-Meta-model to justify the concept of event as basis for
class and object identification. Our meta-model addresses certain issues like what is an
event, in the context of OOAD and why events should be basis to derive static model of a
system (class diagram).

2.4.2 Principle of Event-Meta-model
The proposed Event-Meta-model is based on the principle that events are the core elements
of a system and cause a system to change its state. Overall functionality of a system is a
result of successful execution of a chain of events. Users interact with a system through
events. These events trigger usage in a system (i.e. a Use Case). Using events, analysts can
(a) record changes that have occurred over a period of time; (b) identify which object(s)
have stimulated events, which object(s) have been affected by events, what operations have
made the changes and in which state members. Thus, not only objects but also, changes in
their attributes, invocation of specific operations and relationships among objects can also
be identified by analyzing events and participating objects in events. Values changed in
events, gives an idea of attributes of objects.
This principle of Event Meta-model has formed the basis for our proposed Event-based
methodology for OOA. Next sub section describes the Event Meta-model in detail.

2.4.3 Structure of an Event-Meta-Model
As shown in Figure 2.2, the proposed Event Meta-model has an Event that forms the core
of Meta-model. There are large numbers of events occurring in a system at a given point of
time. Creation and destruction of objects, call to method as well as response from it are all
events. Each event is uniquely identified by giving it a unique Identifier (ID), name and
description which form attributes of an Event in the proposed Event meta-model. The
Meta- Model identifies five types of events – External oriented, Temporal oriented,
**State oriented, Simple event and Complex event.** Complex events are aggregation of Simple events and represents higher level abstraction in an event-based system. E.g, Complex event Reissue article has five simple events reissue requested, checked if overdue, fine collected, check if reserved, article reissued. **System** is another core concept in the proposed Event Meta-model that aggregates various instances of **Object**. This justifies that objects interact and collaborate through events to render the functionality of a system. Every object has a state composed up of **Attributes**. Objects participate in **Relationships** with other objects in or outside system. Object is marked with stereotype “event-actor” that shows objects play role of actors in events. These objects stimulate each other and the stimulus is an event. Object that initiates an event plays the role of an **Initiator**; the one that is affected by an event becomes an **Affecter** and the one that facilitates the occurrence of an event is a **Facilitator**.

There is a **Response** associated with an Event that shows events have a response. Response can be either **Activity** (Use Case) or an **Action**. Activity is a continuous or sequential operation. Activity takes time to complete whereas actions are instantaneous operations associated with an event. Events and Response have a cyclic relationship; an event can trigger a response and a response in turn can generate new events. Information is associated with Response on one side and State on another side which justifies that a response generates information that modifies state of objects (i.e. number of instances, attributes or relationship of an object with other objects).

These elements of Event Meta-Model have formed the basis for defining the structural contents of an event template, which is used for documenting events.
2.5 EVENT TEMPLATES

2.5.1 Proposed Structure of An Event Template

In this section, an Event Template structure is described in detail. Events extracted from textual requirements are documented in Event Templates. An Event Template inherits its components from the proposed Event Meta-Model and models every single interaction details of actors with the system. The components of the proposed Event-Meta-model are mapped to different fields of an Event Template. The important components of an Event Template are:

Event ID: It is a unique alphanumeric value given to each event identified either directly or indirectly from requirements. No two events can be assigned same event id. It helps in tagging detailed description of an event template with its id.
**Event Description:** It is a sentence from requirements that describes an event identified in a Subject-Verb-Object (SVO) pattern. *(e.g. customer places an order)*

**Event Name:** It is a simple name as extracted from requirements given in a natural language. This can indicate verb or verb phrases in a SVO pattern. *(e.g. order placed)*

**Initiator, Facilitator or Affecter:** Initiator starts events. Facilitator facilitates in the occurrence of events and Affecter gets affected as a result of execution of events. Initiator, Facilitator and Affecter are different roles that entities/objects play in different events. Every event has at least one role. An entity can have overlapping roles in events e.g. ‘customer’ can be initiator in one event and affecter in another event. In example, ‘Customer sends complaint through Travel Agency (TA) software’, Travel agency software is a facilitator, Customer is an Initiator and Complaint is an Affecter.

Events occur as a chain of related events. An Event triggers other events in a system. Identification of an event in turn helps to identify other events that can be triggered by it. Such related events of an event are called as Causative events and Trigger vector.

**Causative Events:** Causative events of an event are those events that are reasons behind occurrence of that event. While documenting events, focus is on those causative events that are in context of the problem description. Causative events may not be there for events that are triggered independently after system initialization. As discussed in the previous sections, time for state events cannot be determined so causative events play a very important role in initiating such events.

**Trigger Vector:** It represents a set of events that are triggered as a result of occurrence of an event. An event can trigger either a single event or a set of events that can be executed independently or in parallel. An event relates with other events in an event expression using event operators ‘event-or’, ‘event-and’, ‘event-not’ and ‘event-xor’. Event-or indicates that either none, one or more than one events can be triggered. Event-xor indicates exactly one event can be triggered. Event-and indicates that all events have to be triggered in
parallel. **Event-not** indicates non-occurrence (negation) of an event. For example an event $e_1$ ‘**Customer register with Travel Agency (TA) software**’, triggers an event $e_2$ ‘**Travel agency provides user id and password to Customer**’, $e_1$ is causative event of $e_2$ and $e_2$ is trigger vector of $e_1$. Every event triggered may initiate algorithmically simple or complex services in an object. These services model the behavioral changes in an object. These changes are described in Event Template as Change-event.

**Change-event (state changes):** An event causes operations to be triggered in participating classes. These operations are side effects of any event on the state of participating entities (Initiator, Facilitator or Affector). Operations like creation, termination, and update (or calculate) change the state of participating objects. While operations like read, access, compute or monitor do not change the state. All these operations are described as change-event. E.g. an event ‘**Customer registers with Travel Agency**’ causes a change-event ‘**Creation of Customer Profile object**’. These change-event affect classes at different levels like at (a) Object level change-event can be- Creation (an object is getting created e.g. Order placed), Termination (an object is getting destroyed e.g. Order cancelled), Read objects (an entire object state gets read from an object’s memory). (b) Attribute level change-event can be accessing or updating attributes of objects for performing any calculation, computation and monitoring on that object. An object performs calculations on its value. Monitoring involves checking of an attribute in an object to detect and respond to external systems, devices or other objects. Computation involves computing a functional value from attributes without modifying an object state. (c) Relationship level change-event can be - A classified B that indicates inheritance relationship (e.g Order shipped i.e. Order classified as Shipped Order) or A connected B that indicates association relationship (e.g Person employed i.e. Person is connected/associated to an Organization via ‘is employed’ relationship).

**Timestamp:** Events occur at some point of time. Multiple events may occur at the same time, and could be unrelated, co-operating, or related with each other. Timestamp records time when a particular event has happened or likely to happen in a system. Since all events in a system are related with each other, a relative timestamp value is to be assigned to each
event. Assigning the exact timestamps too early at the analysis level is not possible. So at
the analysis level, dummy timestamp values can be assigned by analysts while identifying
events. A dummy timestamp value can be used in future for reconstructing sequence of
events. Dummy values assigned to timestamp may be fixed or variable in nature. Variable
time stamp indicates that an event occurrence depends on the interaction of user with a
system. Variable timestamp is denoted by unique alphanumeric value starting with ‘TA’
followed by incremental unique id. Higher numerical part of timestamp value indicates a
latter occurrence of that event in a system. For example, an event with variable timestamp
value TA6 will occur earlier in a system in comparison to an event with timestamp value
TA15. Fixed time stamped events indicate periodical events that get initiated after a fixed
interval in a system like weekly, monthly, quarterly etc. Fixed timestamp values are
indicated by Fixed- timestamp daily (FD), Fixed- timestamp weekly (FW), Fixed-
timestamp fortnightly (FF), Fixed- timestamp quarterly (FQ) or Fixed- timestamp annually
(FA). Events with same timestamp value indicate independent events that can occur in
parallel in a system. Timestamp for External and Temporal events could be fixed but for
State events timestamp is always variable, since the time cannot be determined. Different
event templates can be temporally ordered on basis of their timestamp values to map the
flow of activities or steps in a scenario. Our approach has not used timestamps so far,
instead Trigger vector and Causative events are used for ordering events.

**Inputs/Outputs:** Whenever a change event occurs in a system, it requires some inputs or
generates some outputs. Inputs reflect the data needed for a change event whereas output is
the data produced from a change event. Input / Output can contribute to describe attributes
for an object.

**Count:** Count in a template indicates the range (minimum to maximum) of number of
instances of Initiators, Facilitators and Affecters that can participate in an event. Default
value of count is 0..1 (zero or 1). For a given entity count value can be different in different
events. Table 2.1, describes an event template of event “Sensor 1 generates detect signal at
start place”.

39
Table 2.1: Event Template for event “Sensor 1 generates detect signal at start place”

<table>
<thead>
<tr>
<th></th>
<th>Event ID</th>
<th>EA07</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.</td>
<td>Event Name (verb phrase)</td>
<td>Generates detect signal at start place</td>
</tr>
<tr>
<td>3.</td>
<td>Description</td>
<td>Sensor 1 generates detect signal at start place (State/Control Event)</td>
</tr>
<tr>
<td>4.</td>
<td>Initiator</td>
<td>Sensor 1</td>
</tr>
<tr>
<td>5.</td>
<td>Facilitator</td>
<td>ALCS / Belt 1 (Start place)</td>
</tr>
<tr>
<td>6.</td>
<td>Affecter</td>
<td>Signal</td>
</tr>
<tr>
<td>7.</td>
<td>Timestamp</td>
<td>TA20</td>
</tr>
<tr>
<td>8.</td>
<td>Causative Events (Preconditions)</td>
<td>EA05</td>
</tr>
<tr>
<td>9.</td>
<td>Inputs</td>
<td>Signal type</td>
</tr>
<tr>
<td>10.</td>
<td>Trigger Vector</td>
<td>Sensor 2 generates no-detect signal at scan place</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sensor 3 generates no-detect signal at transition place</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sensor 4 generates no-detect signal at end place</td>
</tr>
<tr>
<td>11.</td>
<td>Change-event</td>
<td>Connection between Sensor 1 and Signal</td>
</tr>
</tbody>
</table>

2.5.2 Event Template vs. Use Case Template

The previous sub-section presented different components of an Event Template. This section first gives a detailed description of a Use Case template with an example and then gives a comparison of a Use Case Template with an Event Template to highlight essential differences in the two templates.

2.5.2.1 Use Case Template

Use Case Identification

Use Case ID

Each Use case is given a unique integer sequence number identifier.

Use Case Name

A concise, results-oriented name is given for a Use case. These reflect the tasks a user needs to be able to accomplish using a system. In Use case name, an action verb and a noun is included. For example:

1. View part number information.
2. Manually mark hypertext source and establish link to target.
3. Place an order for a Compact disc (CD) with the updated software version.

**Use Case History**

Use case history records the name of the person who initially documented the Use case along with the date on which the Use case was initially documented. The name of the person who performed the most recent update to the Use case description and along with the date on which the Use case was most recently updated is also provided.

**Use Case Definition**

**Actors**

An actor is a person or other entity external to a software system being specified who interacts with a system and performs use cases to accomplish tasks. Different actors often correspond to different user classes, or roles. An actor who initiates a Use case and participates in completing the Use case is given a name.

**Trigger**

It is the event that initiates the Use case. This could be an external business event or system event that causes the Use case to begin, or it could be the first step in the normal flow.

**Description**

It is a brief description of the reason for and outcome of a Use case, or a high-level description of the sequence of actions and the outcome of executing the Use case.

**Preconditions**

These are any activities that must take place, or any conditions that must be true, before the Use case can be started. Each precondition is given a number. For Examples:

- User’s identity has been authenticated.
- User’s computer has sufficient free memory available to launch task.

**Post-conditions**
These describe the state of a system at the conclusion of a Use case execution. Each post-condition is also given a number. Examples:

2. Price of an item in database has been updated with a new value.

**Normal Flow**

It provides a detailed description of the user actions and system responses that will take place during execution of the Use case under normal, expected conditions. This dialog sequence will ultimately lead to accomplishing the goal stated in the Use case name and description. This description may be written as an answer to the hypothetical question, “How do I <accomplish the task stated in the Use case name>?”. This is best done as a numbered list of actions performed by an actor, alternating with responses provided by a system. The normal flow is numbered “X.0”, where “X” is the Use Case ID.

**Alternative Flows**

Alternative flows document other, legitimate usage scenarios that can take place within a Use case. The alternative flow is stated and any differences in the sequence of steps that take place are described. Each alternative flow is numbered in the form “X.Y”, where “X” is the Use Case ID and Y is a sequence number for the alternative flow. For example, “5.3” would indicate the third alternative flow for Use case number 5.

**Exceptions**

These describe any anticipated error conditions that could occur during execution of a Use case, and define how a system is to respond to those conditions. It also describes how a system is to respond if the Use case execution fails for some unanticipated reason. It is also stated whether the state change is rolled back, completed correctly, partially completed with a known state, or left in an undetermined state as a result of an exception. Each alternative flow is numbered in the form “X.Y.E.Z”, where “X” is the Use Case ID, Y indicates the normal (0) or alternative (>0) flow during which an exception could take place, “E” indicates an
exception, and “Z” is a sequence number for the exceptions. For example “5.0.E.2” would indicate the second exception for the normal flow for Use case number 5.

Includes
This section of the template lists any other Use cases that are included (“called”) by the given Use case. Common functionality that appears in multiple Use cases can be split out into separate Use cases that are included by the ones that need that common functionality.

Priority
It indicates the relative priority of implementing the functionality required to allow a Use case to be executed. The priority scheme used is the same as that used in a software requirements specification.

Frequency of Use
This is an estimate of number of times a Use case will be performed by actors per some appropriate unit of time.

Business Rules
This section list any business rules that influence a Use case.

Special Requirements
This section identifies any additional requirements such as nonfunctional requirements for a Use case that may need to be addressed during design or implementation. These may include performance requirements or other quality attributes.

Assumptions
This section lists any assumptions that were made in the analysis that led to accepting a Use case into the product description and writing the Use case description.

Notes and Issues
This section lists any additional comments about the Use case or any remaining open issues that must be resolved. It identifies who will resolve each issue, the due date, and what the resolution ultimately is.
2.5.2.2 Comparison of event Template and Use Case Template

In this section, a tabular comparison between an Event Template and a Use Case Template fields is presented in Table 2.2 below.

Table 2.2: Comparison of Use Case Template with our Event Template

<table>
<thead>
<tr>
<th>S.No</th>
<th>Template Component</th>
<th>Use Case Template</th>
<th>Event Template</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>ID</td>
<td>√</td>
<td>√</td>
<td>A unique ID will help in tracing, maintaining and relating event templates of all events in the event-based framework.</td>
</tr>
<tr>
<td>2.</td>
<td>Name</td>
<td>√</td>
<td>√</td>
<td>Unlike Use Case template that gives a goal-oriented name; an event name is interaction oriented. The focus is on interactions leading to goals rather than goals in isolation.</td>
</tr>
<tr>
<td>3.</td>
<td>Description</td>
<td>√</td>
<td>√</td>
<td>Unlike a Use Case description that is a sequence of related events; an event description represents a single interaction in a system.</td>
</tr>
<tr>
<td>4.</td>
<td>Actors (Primary and Secondary)</td>
<td>√</td>
<td>Initiator Facilitator Affecter</td>
<td>Unlike a Use Case template that differentiate actors in two category- Primary and Secondary Actor; In an event template, three different roles are defined as an Initiator, a Facilitator or an Affecter of an event. This gives three new stereotypes for classes. Facilitator is optional.</td>
</tr>
<tr>
<td>5.</td>
<td>Timestamp</td>
<td>√</td>
<td>√</td>
<td>Unlike time information that is specified as a non-functional requirement in a Use Case modeling; a timestamp ties an event occurrence with time in a system. It identifies temporal relationships among events; helps to draw an Event Flow Model and can eliminate need for sequence diagram during event-based OOA.</td>
</tr>
<tr>
<td>S.No</td>
<td>Template Component</td>
<td>Use Case Template</td>
<td>Event Template</td>
<td>Comparison</td>
</tr>
<tr>
<td>------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>------------</td>
</tr>
<tr>
<td>6.</td>
<td>Trigger</td>
<td>√</td>
<td>Causative events</td>
<td>Unlike a Use Case where trigger identifies an event that initiates a Use Case; Event Template have causative events which is a set of events that are immediate causes for occurrence of an event.</td>
</tr>
<tr>
<td>7.</td>
<td>Pre-conditions</td>
<td>√</td>
<td>Causative events</td>
<td>Unlike pre-conditions in case of a Use Case Modeling, Event templates have Causative events that must be executed in a system before that event occurs.</td>
</tr>
<tr>
<td>8.</td>
<td>Post-conditions</td>
<td>√</td>
<td>Change-event Trigger-vector</td>
<td>Unlike post-conditions in case of a Use Case Modeling, Event templates have Trigger vector and Change-events to describe side effects in a system due to execution of an event.</td>
</tr>
<tr>
<td>9.</td>
<td>Trigger-vector</td>
<td>×</td>
<td>√</td>
<td>Trigger vector represents event ID’s of events that are triggered / caused due to occurrence of an event.</td>
</tr>
<tr>
<td>10.</td>
<td>Inputs/Outputs</td>
<td>×</td>
<td>√</td>
<td>It represents data that provides input or carries output. Details include entity name and the name of the state members (attributes) and value (content) involved in execution of an event.</td>
</tr>
<tr>
<td>11.</td>
<td>Change-event</td>
<td>×</td>
<td>√</td>
<td>Changes in a system are categorised in terms of 13 different Change events as described in Event Template. Every change in an event template records the method-name, its types (event category) and which class realises it.</td>
</tr>
<tr>
<td>12.</td>
<td>Normal Flow/Alternative Flows</td>
<td>√</td>
<td>Event Flow Diagram</td>
<td>Unlike normal and alternative flows in a Use Case template, Event flow is determined either through causal or temporal ordering of the event templates based on timestamp value of each event or using causative and trigger vector of each event template.</td>
</tr>
</tbody>
</table>
2.6 SUMMARY

This chapter has presented the objective and principle behind the proposed event-based framework. It has introduced the concept of events, their types and significance. The proposed Event Meta-model, its principle and role in the proposed Event-based framework; and Event Templates for documenting events is explained. The meta-model has addressed certain issues like what an event is, in the context of OOAD and why events should be basis to derive static model of the system (class diagram).

In the light of critical review of Use Case based approaches presented in Chapter 1, a detailed comparison is also presented of an Event Template with a Use Case Template, with an aim to clarify the role and importance of Event Template in Event-based OOA. In the next chapter, we will discuss the process to model requirements based on analysis of events occurring in the system and to generate Class diagram specification from Event templates. A controlled experiment will also be described to evaluate and compare the proposed event-based approach to build Class diagrams with a more conventional and industry standard Use Case based approach.