**OBJECT ORIENTED ANALYSIS AND DESIGN**

**UNIT 4**

**EVENTS AND SIGNALS**

Introduction

In state machines (sequence of states), we use events to model the occurrence of a stimulus that can trigger an object to move from one state to another state. Events may include signals, calls, the passage of time or a change in state.

In UML, each thing that happens is modelled as an event. An event is the specification of a significant occurrence that has a location in time and space. A signal, passing of time and change in state are asynchronous events. Calls are generally synchronous events, representing invocation of an operation.

UML allows us to represent events graphically as shown below. Signals may be represented as stereotyped classes and other events are represented as messages associated with transitions which cause an object to move from one state to another.

Types of Events

Events may be external or internal. Events passed between the system and its actors are external events. For example, in an ATM system, pushing a button or inserting a card are external events. Internal events are those that are passed among objects living inside the system. For example, a overflow exception generated by an object is an internal event.

In UML, we can model four kinds of events namely: signals, calls, passing of time and change in state.

**Signals**

 A signal is a named object that is sent asynchronously by one object and then received by another. Exceptions are the famous examples for signals. A signal may be sent as the action of a state in a state machine or as a message in an interaction. The execution of an operation can also send signals.

In UML, we model the relationship between an operation and the events using a dependency stereotyped with “send”, which indicates that an operation sends a particular signal.

**Call Events**

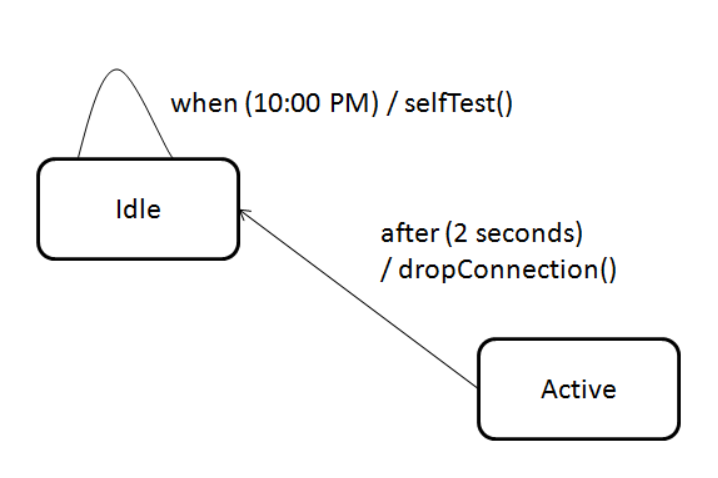
A call event represents the dispatch of an operation from one object to another. A call event may trigger a state change in a state machine. A call event, in general, is synchronous.

This means that the sender object must wait until it gets an acknowledgment from the receiver object which receives the call event. For example, consider the states of a customer in an ATM application:

**Time and Change Events**

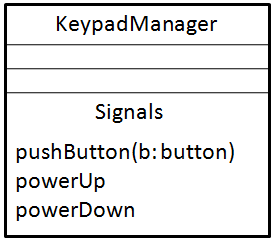
 A time event represents the passage of time. In UML, we model the time event using the “after” keyword followed by an expression that evaluates a period of time.

A change event represents an event that represents a change in state or the satisfaction of some condition. In UML, change event is modeled using the keyword “when” followed by some Boolean expression.

**Sending and Receiving Events**

Any instance of a class can receive a call event or signal. If this is a synchronous call event, the sender is in locked state with receiver. If this is a signal, then the sender is free to carry its operations without any concern on the receiver.

In UML, call events are modeled as operations on the class of an object and signals that an object can receive are stored in an extra component in the class as shown below:

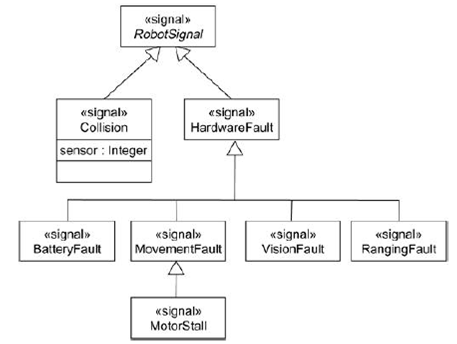
[](http://www.startertutorials.com/uml/wp-content/uploads/2013/10/5-signals-representation.gif)

**Common Modeling Techniques**

**Modeling a family of signals**

 To model a family of signals,

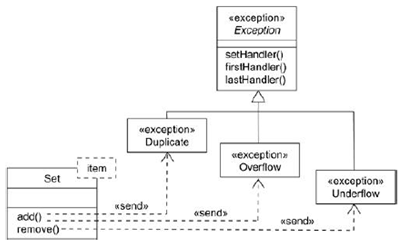
1. Consider all the signals to which a set of objects can respond.
2. Arrange these signals in a hierarchy using generalization-specialization relationship.
3. Look out for polymorphism in the state machine of the active objects. When polymorphism is found, adjust the hierarchy by introducing intermediate abstract signals.

[](http://www.startertutorials.com/uml/wp-content/uploads/2013/10/6-modeling-signals.gif)

### **Modeling Exceptions**

 To model exceptions,

1. For each class and interface and for each operation of such elements, consider the exceptional conditions that might arise.
2. Arrange these exceptions in a hierarchy.
3. For each operation, specify the exceptions that it may rise.

[](http://www.startertutorials.com/uml/wp-content/uploads/2013/10/7-modeling-exceptions.gif)

**Processes and Threads**

* A process is a heavyweight flow that can execute concurrently with other processes.
* A thread is a lightweight flow that can execute concurrently with other threads within the same process.
* An active object is an object that owns a process or thread and can initiate control activity.
* An active class is a class whose instances are active objects.
* Graphically, an active class is rendered as a rectangle with thick lines. Processes and threads are rendered as stereotyped active classes. Figure 1:Active class

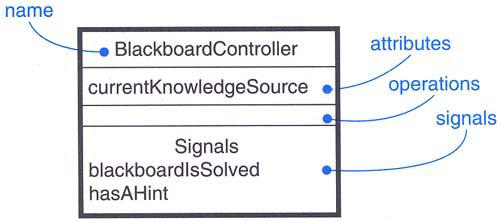
[](https://praveenthomasln.files.wordpress.com/2012/04/figure-1-active-class.png)

Figure 1: Active Class

**Flow of Control**  
*In a sequential system*, there is a single flow of control. i.e, one thing, and one thing only, can take place at a time.  
*In a concurrent system*, there is multiple simultaneous flow of control i.e, more than one thing can take place at a time.

**Classes and Events**

* Active classes are just classes which represents an independent flow of control
* Active classes share the same properties as all other classes.
* When an active object is created, the associated flow of control is started; when the active object is destroyed, the associated flow of control is terminated
* *two standard stereotypes* that apply to active classes are, ***<<process>>*** – Specifies a heavyweight flow that can execute concurrently with other processes. (heavyweight means, a thing known to the OS itself and runs in an independent address space) ***<<thread>>*** – Specifies a lightweight flow that can execute concurrently with other threads within the same process (lightweight means, known to the OS itself.)
* All the threads that live in the context of a process are peers of one another

**Communication**

* In a system with both active and passive objects, there are *four possible combinations of interaction*
* *First*, a message may be passed from one passive object to another
* *Second*, a message may be passed from one active object to another
* In *inter-process communication* there are two possible styles of communication. *First*, one active object might synchronously call an operation of another. *Second*, one active object might asynchronously send a signal or call an operation of another object
* a synchronous message is rendered as a full arrow and an asynchronous message is rendered as a half arrow
* Figure 2: shows Communication
* *Third*, a message may be passed from an active object to a passive object
* *Fourth*, a message may be passed from a passive object to an active one

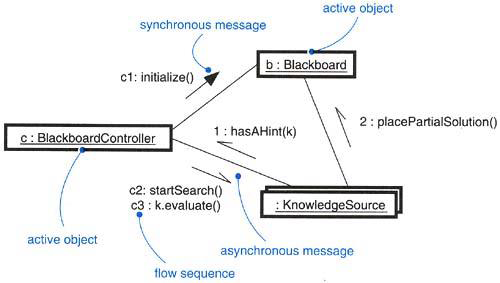
[](https://praveenthomasln.files.wordpress.com/2012/04/figure-2-communication.png)

Figure 2: Communication

Advertisements

REPORT THIS ADPRIVACY

**Synchronization**

* synchronization means arranging the flow of controls of objects so that mutual exclusion will be guaranteed.
* in object-oriented systems these objects are treated as a critical region
* Figure 3 Synchronization
* *three approaches* are there to handle synchronization:
* *Sequential* – Callers must coordinate outside the object so that only one flow is in the object at a time
* *Guarded* – multiple flow of control is sequentialized with the help of object’s guarded operations. in effect it becomes sequential.
* *Concurrent* – multiple flow of control is guaranteed by treating each operation as atomic
* synchronization are rendered in the operations of active classes with the help of constraints

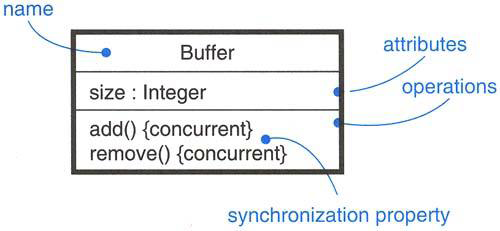
[](https://praveenthomasln.files.wordpress.com/2012/04/figure-3-synchronization.png)

Figure 3: Synchronization

**Process Views**

* The process view of a system encompasses the threads and processes that form the system’s concurrency and synchronization mechanisms.
* This view primarily addresses the performance, scalability, and throughput of the system.

**Modeling Multiple Flows of Control**  
To model multiple flows of control,

* · Identify the opportunities for concurrent action and reify each flow as an active class. Generalize common sets of active objects into an active class. Be careful not to overengineer the process view of your system by introducing too much concurrency.
* · Consider a balanced distribution of responsibilities among these active classes, then examine the other active and passive classes with which each collaborates statically. Ensure that each active class is both tightly cohesive and loosely coupled relative to these neighboring classes and that each has the right set of attributes, operations, and signals.
* · Capture these static decisions in class diagrams, explicitly highlighting each active class.
* · Consider how each group of classes collaborates with one another dynamically. Capture those decisions in interaction diagrams. Explicitly show active objects as the root of such flows. Identify each related sequence by identifying it with the name of the active object.
* · Pay close attention to communication among active objects. Apply synchronous and asynchronous messaging, as appropriate.
* · Pay close attention to synchronization among these active objects and the passive objects with which they collaborate. Apply sequential, guarded, or concurrent operation semantics, as appropriate.

Advertisements

REPORT THIS ADPRIVACY

Figure 4 shows part of the process view of a trading system.

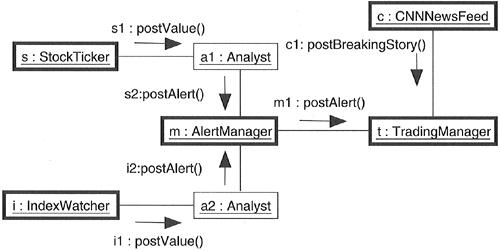
[](https://praveenthomasln.files.wordpress.com/2012/04/figure-4-modeling-flows-of-control.png)

Figure 4: Modeling Flows of Control

**Modeling Interprocess Communication**  
To model interprocess communication,

* · Model the multiple flows of control.
* · Consider which of these active objects represent processes and which represent threads. Distinguish them using the appropriate stereotype.
* · Model messaging using asynchronous communication; model remote procedure calls using synchronous communication.
* · Informally specify the underlying mechanism for communication by using notes, or more formally by using collaborations.

Figure 5 shows a distributed reservation system with processes spread across four nodes.

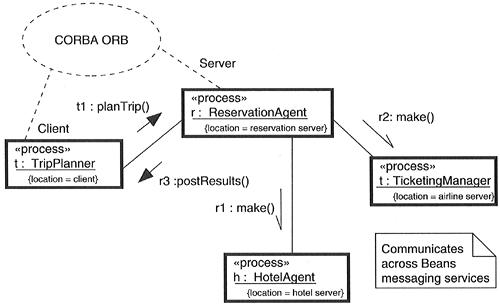
[](https://praveenthomasln.files.wordpress.com/2012/04/figure-5-modeling-interprocess-communication.png)

Figure 5: Modeling Interprocess Communication