**INTRODUCTION**

**UML**

* Unified Modeling Language (UML) is a standard language for creating blueprints that depict the structure and design of the software system.
* Unified Modeling Language (UML) is a language for specifying, constructing, visualizing and documenting the artifacts of a software-intensive system.
* Tools: Rational Rose, Jude, ArgoUMl, Visio and Poseidon.
* Provide notations for classes, objects and interfaces
* Allows you to create diagram to visualize a software system
* Allows code generation from a UML model
* You can use the diagrams as input documents for the subsequent phases of SDLC

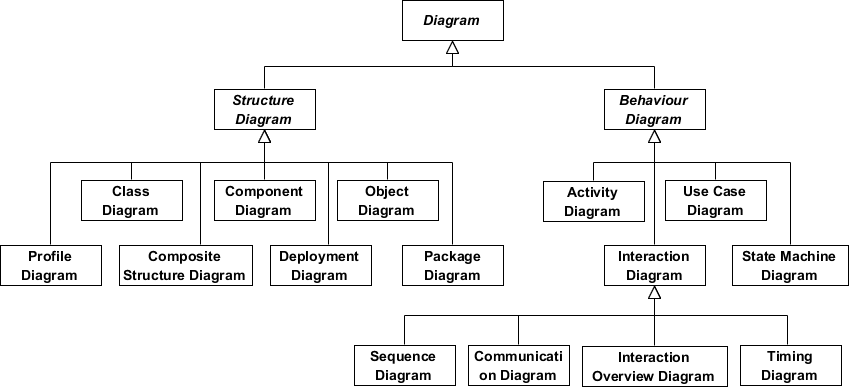
**Types of UML diagrams**

**Structure diagrams**

* Structure diagrams show the things in the modeled system.
* They show different objects in a system.

**Behavioral diagrams**

* Behavioral diagrams show what should happen in a system.
* They describe how the objects interact with each other to create a functioning system.



**USE CASE DIAGRAM**

* It depicts the various operations that a system performs.
* It contains
  + use cases: sequence of actions that form a single unit of work for an actor
  + Actor: a user who is external to the system and interacts with the use case
  + Relationships: an interaction that is possible between the actor and use cases
* The use case confine who does what with the system, for what purpose, without dealing with system internals.
* **Purpose:**
  + Documenting existing process
  + Analyzing new process concepts
  + Identifying IT levers, if any
  + Finding out re-engineering opportunities
  + Identifying the boundaries of the system by making a semantic network diagram
  + Changing and extending the actor’s or user’s functionality
  + Addressing the non-functional requirements for some scenarios
* **Guideline to draw**
  + Be familiar with your business
  + Keep matters abstract
  + Requirement specification with creativity and visions
  + Avoid functional decomposition

**Notation**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME** | **SYMBOL** | **DESCRIPTION** |
| 1 | Actor |  | Internal or external entity that interacts with the system.  Name by noun |
| 2 | Use case |  | Use case is used to capture high level functionalities of a system.  Named by verb + Noun (or Noun Phrase). |
| 3 | Dependency |  | A semantic relationship between two elements in which a change to one element may affect the meaning of other |
| 4 | Association |  | A structural relationship describing a set of links connected between objects. |
| 5 | Generalization |  | A relationship in which objects of a specialized element (child) are substitutable for objects of a generalized element (parent). |

**Example**

A close up of a map

Description automatically generated

**CLASS DIAGRAM**

* A class diagram represents a set of classes, interface and their relationships.
* It is represented by a rectangular box with three compartments.
  + Class name
  + Attributes of the class
  + Methods associated with the class
* **Purpose:**
  + Analysis and design of the static view of an application.
  + Describe responsibilities of a system.
  + Base for component and deployment diagrams.
  + Forward and reverse engineering.
* **Guideline to draw**
  + The name of the class diagram should be meaningful to describe the aspect of the system.
  + Each element and their relationships should be identified in advance.
  + Responsibility (attributes and methods) of each class should be clearly identified.
  + For each class minimum number of properties should be specified. Because unnecessary properties will make the diagram complicated.
  + Use notes when ever required to describe some aspect of the diagram. Because at the end of the drawing it should be understandable to the developer/coder.
  + Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.
* **Uses:**
  + Describing the static view of the system.
  + Showing the collaboration among the elements of the static view.
  + Describing the functionalities performed by the system.
  + Construction of software applications using object oriented languages.

**Notation**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME** | **SYMBOL** | **DESCRIPTION** |
| 1 | Dependency |  | A semantic relationship between two elements in which a change to one element may affect the meaning of other |
| 2 | Association |  | A structural relationship describing a set of links connected between objects. |
| 3 | Generalization |  | A relationship in which objects of a specialized element (child) are substitutable for objects of a generalized element (parent). |
| 4 | Class |  | The top section is used to name the class.The second one is used to show the attributes of the class.The third section is used to describe the operations performed by the class. |
| 5 | Aggregation |  | It is a special type of association depicting whole part relationship. |
| 6 | Composition |  | Composition is a special type of aggregation that denotes a strong ownership between Classes, the whole and its part. |
| 7 | Relationship Names |  | Names of relationships are written in the middle of the association line.  Good relation names make sense when you read them out loud: |
| 8 | Member access modifiers | * Public (+) * Private (-) * Protected (#) * Package (~) * Derived (/) * Static (underlined) | All classes have different access levels depending on the access modifier (visibility) |
| 9 | Multiplicity |  | How many objects of each class take part in the relationships |

**Example**

A close up of text on a white background

Description automatically generated

**SEQUENCE DIAGRAM**

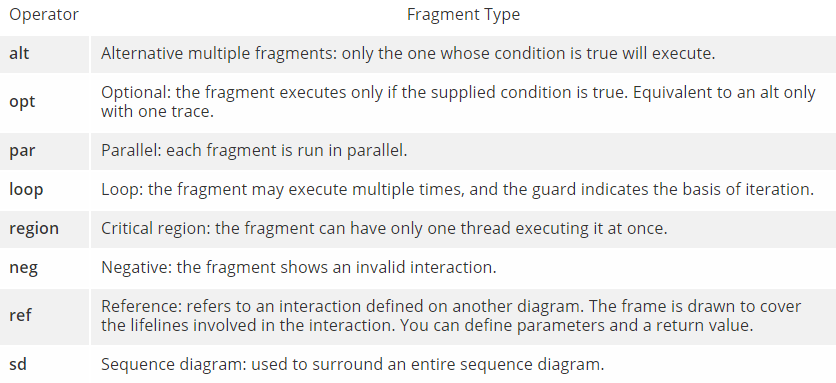
* Sequence diagram represent an interaction between object in the form of messages ordered in a sequence by time.
* The difference between the sequence and communication diagram is that communication diagram emphasize on the structural organization of objects as opposed to sequence diagram that show the messages exchanged between objects ordered in a sequence by time.
* We can draw a sequence diagram for any given system by using the classes and the use cases identified for the system.
* **Purpose:**
  + To capture dynamic behaviour of a system.
  + To describe the message flow in the system.
  + To describe structural organization of the objects.
  + To describe interaction among objects.
* **Guideline to draw:**

The following things are to identified clearly before drawing the sequence diagram:

* + Objects taking part in the interaction.
  + Message flows among the objects.
  + The sequence in which the messages are flowing.
  + Object organization.
* **Uses:**
  + To model flow of control by time sequence.
  + To model flow of control by structural organizations.
  + For forward engineering.
  + For reverse engineering.

**Notation**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME** | **SYMBOL** | **DESCRIPTION** |
| 1 | Object |  | A classifier describes a set of instances that have common behavioral and structural features. |
| 2 | Actor |  | A lifeline notation with an actor element symbol is used when the particular sequence diagram is owned by a use case. |
| 3 | Entity |  | A lifeline with an entity element represents system data. For example, in a customer service application, the Customer entity would manage all data related to a customer. |
| 4 | Boundary |  | A lifeline with a boundary element indicates a system boundary/ software element in a system; for example, user interface screens, database gateways or menus that users interact with, are boundaries. |
| 5 | Control |  | lifeline with a control element indicates a controlling entity or manager. It organizes and schedules the interactions between the boundaries and entities and serves as the mediator between them. |
| 6 | Lifeline symbol |  | Represents the passage of time as it extends downward. This dashed vertical line shows the sequential events that occur to an object during the charted process. Lifelines may begin with a labeled rectangle shape or an actor symbol. |
| 7 | Activation Box |  | Represents the time needed for an object to complete a task. The longer the task will take, the longer the activation box becomes. |
| 8 | Synchronous message/Call action |  | This action send message and wait for a reply. The reply message is send back to that message. |
| 9 | Reply message |  | Represented by a dashed line with a lined arrowhead, these messages are replies to calls. |
| 10 | Asynchronous message |  | Send message and proceed immediately without waiting for return value. |
| 11 | Create message/ constructor message |  | Represented by a dashed line with a lined arrowhead. This message creates a new object. |
| 12 | Delete message / Destructor message |  | Represented by a solid line with a solid arrowhead, followed by an X. This message destroys an object. |
| 13 | Self message |  | A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself. |
| 14 | Sequence fragments | Fragment | Sequence fragments make it easier to create and maintain accurate sequence diagrams  A sequence fragment is represented as a box, called a combined fragment, which encloses a portion of the interactions within a sequence diagram  The fragment operator (in the top left cornet) indicates the type of fragment  Fragment types: ref, assert, loop, break, alt, opt, neg |

****

**Example**

A close up of a map

Description automatically generated

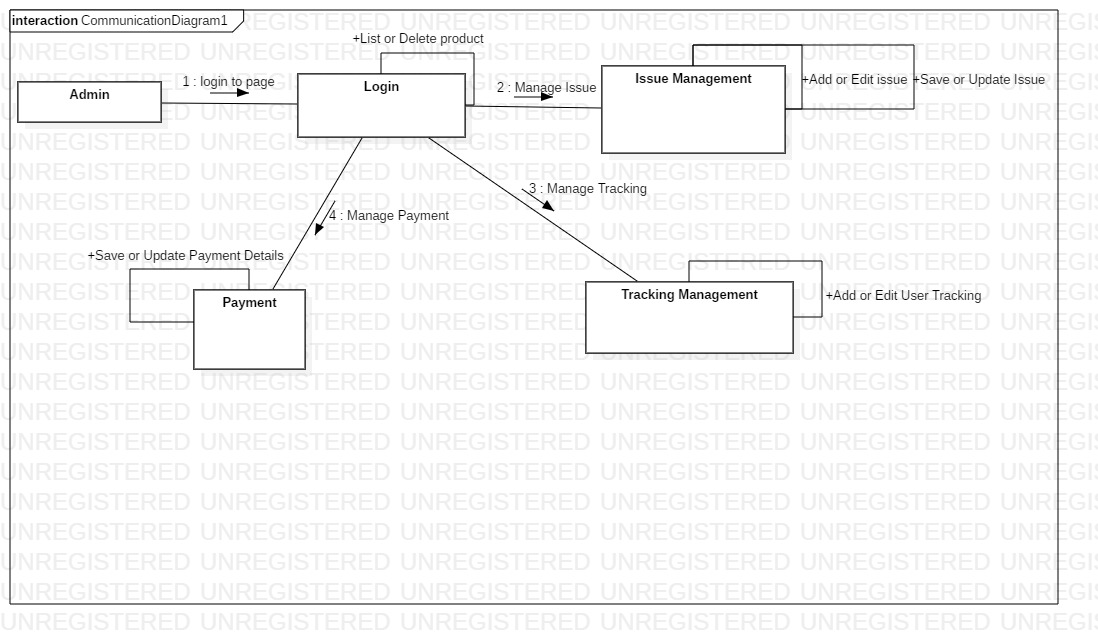
**COMMUNICATION DIAGRAM/ COLLABORATION DIAGRAM**

* Communication diagram represent an interaction between objects in the form of message.
* Communication diagram are also called as collaboration diagram.
* **Purpose**
  + A collaboration diagram shows the objects and relationships involved in an interaction, and the sequence of messages exchanged among the objects during the interaction.
* **Uses:**
  + Collaboration diagrams are used to show how objects interact to perform the behavior of a particular use case, or a part of a use case.
  + Along with sequence diagrams, collaborations are used by designers to define and clarify the roles of the objects that perform a particular flow of events of a use case.

**Notation**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME** | **SYMBOL** | **DESCRIPTION** |
| 1 | Object / Instance | http://ics.upjs.sk/~novotnyr/home/skola/softverove_inzinierstvo/uml%20(telelogic)/images/dgmsuml64.gif | An instance in a collaboration diagram represents an instantiation of a class in a class diagram or a use case in a use case diagram |
| 2 | Actor | http://ics.upjs.sk/~novotnyr/home/skola/softverove_inzinierstvo/uml%20(telelogic)/images/dgmsuml70.gif | An actor in a collaboration diagram represents the person, software, hardware, or other agent external to the system that is interacting with the system. |
| 3 | Frame | Interaction Frame for Communication Diagram BuyItem. | Communication diagrams could be shown within a rectangular **frame** with the **name** in a compartment in the upper left corner. |
| 4 | Message | * An arrow pointing from the commencing object to the destination object shows the interaction between the objects. * The number represents the order/sequence of this interaction.   **Message direction**   * There are two message directions: * Backward Messages * Forward Messages * The direction in which the link was drawn determines the direction of the message.   **Message type**   * You can create these types of messages: * [Nested message](http://ics.upjs.sk/~novotnyr/home/skola/softverove_inzinierstvo/uml%20(telelogic)/dgmsuml3.html#1055623) * [Flat message](http://ics.upjs.sk/~novotnyr/home/skola/softverove_inzinierstvo/uml%20(telelogic)/dgmsuml3.html#1040681) * [Asynchronous message](http://ics.upjs.sk/~novotnyr/home/skola/softverove_inzinierstvo/uml%20(telelogic)/dgmsuml3.html#1154039) | |
| 5 | Nested Message |  | The nested message represents a procedure call or other nested flow of control. The nested sequence is completed before the outer level sequence resumes. |
| 6 | Flat message |  | The flat message shows the progression to the next step in a sequence. |
| 7 | Asynchronous message |  | The asynchronous message symbol shows an asynchronous message between two objects. |

**Example**

****

**ACTIVITY DIAGRAM**

* Activities are a representation of the various operations performed by a class.
* An activity diagram depicts the flow of control from one activity to another.
* We can draw an activity diagram by identifying the activities performed by the various classes of the system.
* **Purpose:**
  + Draw the activity flow of a system.
  + Describe the sequence from one activity to another.
  + Describe the parallel, branched and concurrent flow of the system.
* **Guideline to draw:**

Before drawing an activity diagram we should identify the following elements:

* + Activities
  + Association
  + Conditions
  + Constraints
* **Uses :**
  + Modeling work flow by using activities.
  + Modeling business requirements.
  + High level understanding of the system's functionalities.
  + Investigate business requirements at a later stage.

**Notation**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME** | **SYMBOL** | **DESCRIPTION** |
| 1 | Initial state |  | It shows the starting point of a process. |
| 2 | Final state |  | It shows the end of all flows in an activity |
| 3 | Flow Final State |  | It shows the end of all flows in that instance |
| 4 | Choice |  | Collection of operation that specifies a service of class. |
| 5 | Junction/ Decision making/Guard |  | A diamond represents a decision with alternate paths.  Guards are a statement written next to a decision diamond that must be true before moving next to the next activity. |
| 6 | Fork |  | A synchronization bar helps illustrate parallel transitions  Split behavior into a set of parallel or concurrent flows of activities |
| 7 | Join |  | A synchronization bar helps illustrate parallel transitions  Bring back together a set of parallel or concurrent flows of activities |
| 8 | Action |  | Action states represent the noninterruptible actions of objects.  A task to be performed |
| 9 | Swimlane and Partition | Activity Diagram Notation - Swimlane and Partition | A way to group activities performed by the same actor on an activity diagram or to group activities in a single thread |
| 10 | Time Event |  | This refers to an event that stops the flow for a time; an hourglass depicts it. |
| 11 | Merge Event |  | A merge event brings together multiple flows that are not concurrent. |
| 12 | Sent and Received Signals |  | Signals represent how activities can be modified from outside the system. They usually appear in pairs of sent and received signals, because the state can't change until a response is received, much like synchronous messages in a sequence diagram. |
| 13 | Interrupting Edge |  | An event, such as a cancellation, that interrupts the flow denoted with a lightning bolt. |

**Example**

A screen shot of a computer

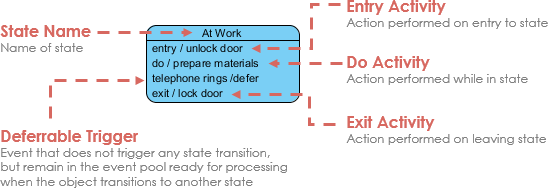
Description automatically generated

**STATE MACHINE DIAGRAM**

### What is a State?

"A state is an abstraction of the attribute values and links of an object. Sets of values are grouped together into a state according to properties that affect the gross behavior of the object."

### State Notation



* A state machine diagram shows how a class reacts when an event occurs.
* We can draw a state machine diagram for any given system by using the classes and the use cases identified for the system.
* Statechart diagrams are also called as state machine diagrams.
* These diagrams are used to model the event-based system.
* A state of an entity is controlled with the help of an event.
* There is a total of two types of state machine diagrams: 1) Behavioral 2) State machine 3) Protocol state machine
* Statechart diagram is used to capture the dynamic aspect of a system.
* A state is a specific moment in the lifespan of an object.

Unified Modeling Language defines three types of states:

* **Simple state**
  + They do not have any substrate.
* **Composite state**
  + These types of states can have one or more than one substate.
  + A composite state with two or more substates is called an orthogonal state.
* **Submachine state**
  + These states are semantically equal to the composite states.
  + Unlike the composite state, we can reuse the submachine states.
* **Purpose:**
  + To model dynamic aspect of a system.
  + To model life time of a reactive system.
  + To describe different states of an object during its life time.
  + Define a state machine to model states of an object.
* **Guideline to draw:**

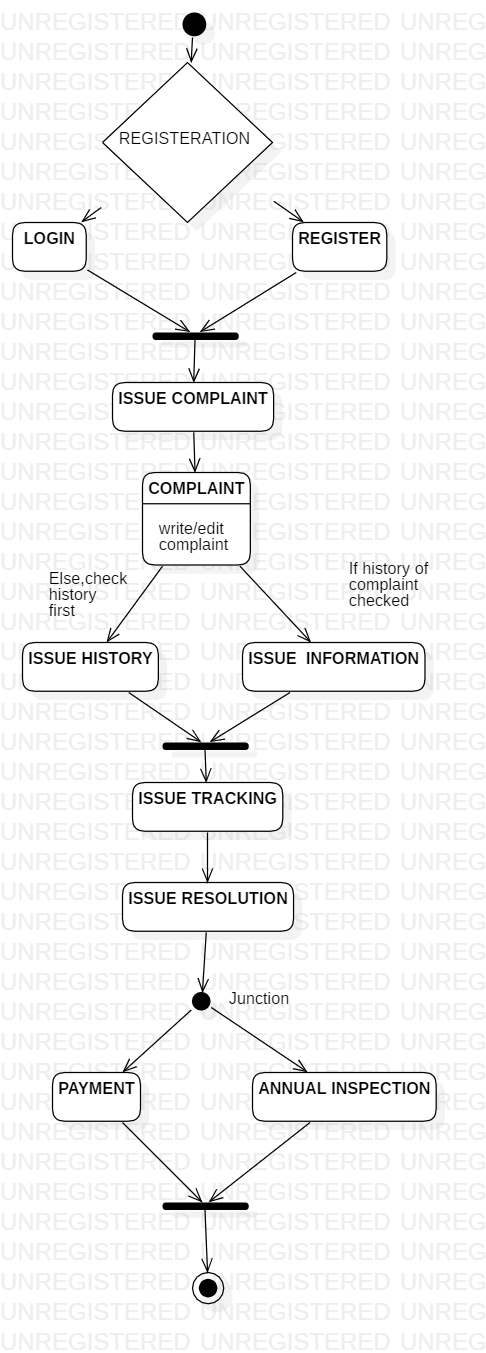
Before drawing a Statechart diagram we must have clarified the following points:

* + Identify important objects to be analyzed.
  + Identify the states.
  + Identify the events.
* **Uses:**
  + To model object states of a system.
  + To model reactive system. Reactive system consists of reactive objects.
  + To identify events responsible for state changes.
  + Forward and reverse engineering.

**Notation:**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME** | **SYMBOL** | **DESCRIPTION** |
| 1 | Initial state |  | It shows the starting point of a process. |
| 2 | Final state |  | It shows the end of a process. |
| 3 | Transition |  | path between different states of an object.  A state can have a transition that points back to itself. |
| 4 | Entry Point |  | Collection of operation that specifies a service of class. |
| 5 | Exit Point |  | The point at which an object escapes the composite state or state machine. The exit point is typically used if the process is not completed but has to be escaped for some error or other issue. |
| 6 | Junction |  | A junction point indicates that several states can transition to the same state on a given event |
| 7 | Terminate |  | Used for interrupt |
| 8 | Choice |  | A diamond represents a decision with alternate paths. |
| 9 | Fork |  | A synchronization bar helps illustrate parallel transitions |
| 10 | Join |  | A synchronization bar helps illustrate parallel transitions |
| 11 | Simple State |  | A state represents the conditions or circumstances of an object of a class at an instant of time. |
| 12 | Composite State |  | States represent situations during the life of an object  These types of states can have one or more than one substate. |
| 13 | Submachine state |  | These states are semantically equal to the composite states.  Unlike the composite state, we can reuse the submachine states. |
| 14 | Orthogonal State |  | A composite state with two or more substates is called an orthogonal state |
| 15 | Shallow History |  | History refers to the development of object-oriented methods and notation.  History states allow the state machine to re-enter the last substate that was active prior to leaving the composite state. |
| 16 | Deep History |  | Deep-history is similar to shallow-history except it represents the most recent leaf-substate of the composite state. |
| 17 | Frame |  | It is used for reference of state machine |

**Example**

****

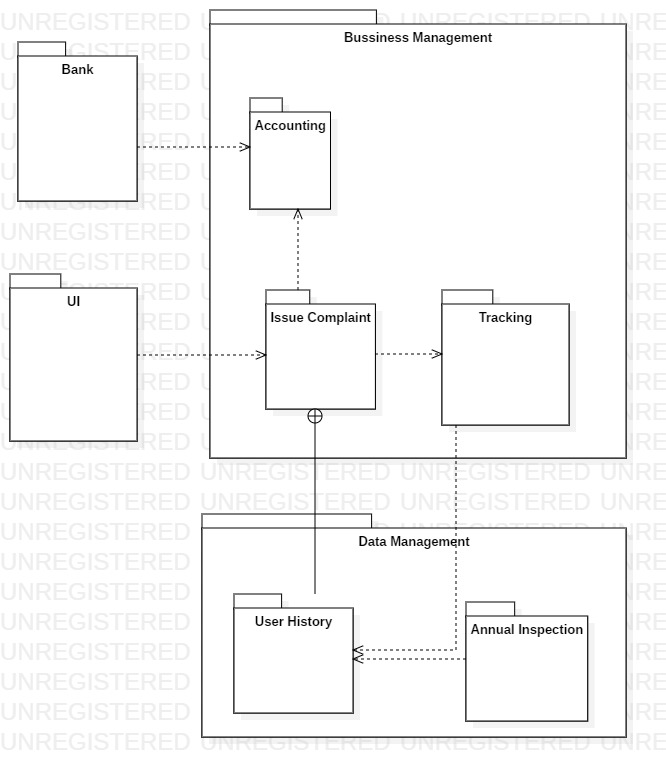
**PACKAGE DIAGRAM**

* All the interrelated classes and interfaces of the system when grouped together form a package.
* To represent all these interrelated classes and interface UML provides package diagram.
* Package diagram helps in representing the various packages of a software system and the dependencies between them.
* It also gives a high-level impression of use case and class diagram.
* **Purpose**
  + To provide static models of modules, their parts and their relationships
  + To present the architectural modelling of the system
  + To group any UML elements
  + To specify the logical distribution of classes
  + To emphasize the logical structure of the system
  + To offer the logical distribution of classes which is inferred from the logical architecture of the system
* **Uses**
  + To illustrate the functionality of a software system.
  + To illustrate the layered architecture of a software system.
  + The dependencies between these packages can be adorned with labels / stereotypes to indicate the communication mechanism between the layers.

**Notation**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME** | **SYMBOL** | **DESCRIPTION** |
| 1 | Package |  | Organize elements into groups to provide better structure for system model. |
| 2 | Model |  | Show only a subset of the contained elements according to some criterion. |
| 3 | Subsystem |  | You can model an entire system as a hierarchy of subsystems. |
| 4 | Containment |  | This notation for packages is semantically equivalent to [composition](https://www.uml-diagrams.org/composition.html) |
| 5 | Dependency |  | <<import>> - one package imports the functionality of other package  <<access>> - one package requires help from functions of other package. |

**Example**



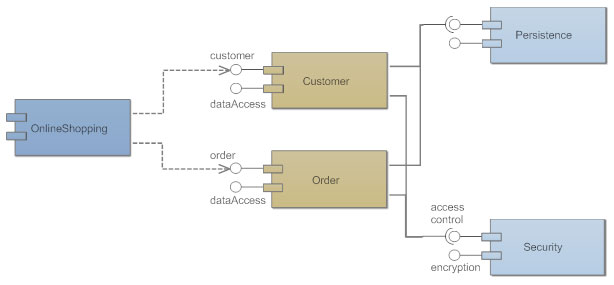
**COMPONENT DIAGRAM**

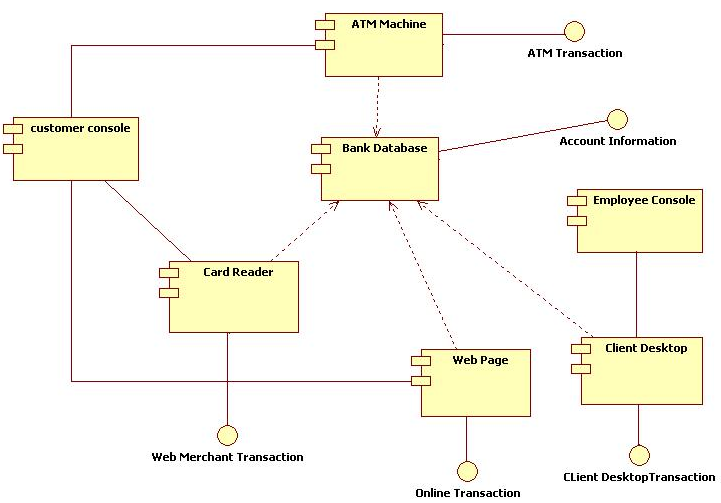
* A component diagram shows the physical view of the system
* A component is an autonomous unit within a system.
* We combine packages or individual entities to form components.
* We can depict various components and their dependencies using a component diagram.
* Component diagram contain: component package, components, interfaces and dependency relationship.
* **Purpose:**
  + It shows the structural relationship between the components of a system.
  + It identifies the architectural perspective of the system as they enable the designer to model the high level software components with their interfaces to other components.
  + It helps to organize source code into manageable chunks called components.
  + It helps to specify a physical database.
  + It can be easily developed by architects and programmers.
  + It enables to model the high level software components and the interfaces to those components.
  + The components and subsystem can be flexibly reused and replaced.
* **Guideline to draw**
  + Based on the analysis of the problem description of the system, identify the major subsystem.
  + Group the individual packages and other logical entities in the system to provide as separate components.
  + Then identify the interfaces needed for components interaction.
  + If needed, identify the subprograms which are part of each of the components and draw them along with their associated components.
  + Use appropriate notations to draw the complete component diagram.

**Notation**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME** | **SYMBOL** | **DESCRIPTION** |
| 1 | Component |  | Component is used to represent any part of a system for which UML diagrams are made.  A module of classes that represent independent systems or subsystems with the ability to interface with the rest of the system. |
| 2 | Connector |  | A link that enables communication between two or more instances playing some roles  Types :   * [**Delegation connector**](https://www.uml-diagrams.org/composite-structure-diagrams/connector.html#delegation-connector) : connector that links the external contract of a component to the realization of that behavior. * [**Assembly connector**](https://www.uml-diagrams.org/composite-structure-diagrams/connector.html#assembly-connector): a connector between two or more parts or ports |
| 3 | Port |  | Specifies a separate interaction point between the component and the environment. |
| 4 | Interface |  | **Provided interfaces**  symbols with a complete circle  the interfaces where a component produces information used by the required interface of another component.  **Required interfaces**  symbols with only a half circle  the interfaces where a component requires information in order to perform its proper function. |
| 5 | Dependency |  | A dependency is a relationship that signifies that a single or a set of model elements requires other model elements for their specification or implementation. |
| 6 | Realization |  | **Interface Realization:** an interface realization relationship is a specialized type of implementation relationship between a classifier and a provided interface.  **Component Realization**:specialized [realization](https://www.uml-diagrams.org/realization.html) dependency used to (optionally) define [classifiers](https://www.uml-diagrams.org/classifier.html) that realize the contract offered by a component in terms of its [provided interfaces](https://www.uml-diagrams.org/class.html#provided-interface) and [required interfaces](https://www.uml-diagrams.org/component.html#required-interface). |

**Example**





**DEPLOYMENT DIAGRAM**

* A deployment diagram shows the physical placement of components in nodes over a network.
* A deployment diagram can be drawn by identifying nodes and components.
* A deployment diagram usually describes the resources required for processing and the installation of software components in those resources.
* **Purpose**
  + It shows the relationship between software and hardware components in the target system.
  + They are useful to show the system design that has subsystem, concurrent execution, compile time and execution time invocations, and hardware/software mapping by assigning the appropriate software components to the hardware devices.
  + As they specify the distribution of software components in various devices and processors in the target environment, it will be easier for maintenance activities.
  + Using this diagram it is easier to identify performance bottlenecks.
* **Guideline to draw**
  + Identify the hardware components and processing units in the target system.
  + Analyze the software and find out the subsystem, parallel execution of modules, server side components, client side components, business logic components, backend database servers and software and hardware mapping mechanism to map the software components to be mapped with appropriate hardware devices.
  + Draw the hardware components and show the software components inside them and also show the connectivity between them.

**Notation**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME** | **SYMBOL** | **DESCRIPTION** |
| 1 | Node |  | A node represents a physical component of the system. Node is used to represent physical part of a system like server, network etc.  Represents hardware or software objects, which are of a higher level than components. |
| 2 | Association |  | A structural relationship describing a set of links connected between objects. |

**Example**

