SYSTEM DESIGN

3. SYSTEM DESIGN

Once the analysis stage is completed, the next stage is to determine in broad outline form how the problem might be solved. During system design, we are beginning to move from the logical to physical level.

System design involves architectural and detailed design of the system. Architectural design involves identifying software components, decomposing them into processing modules and conceptual data structures, and specifying the interconnections among components.

Detailed design is concerned with how to package processing modules and how to implement the processing algorithms, data structures and interconnections of standard algorithms, invention of new algorithms, and design of data representations and packaging of software products. Two kinds of approaches are available: When embarking on system design, it's imperative to steer clear of certain pitfalls that could compromise the effectiveness and efficiency of the solution. One of the most common mistakes is succumbing to over-engineering, wherein unnecessary complexity is introduced, leading to bloated systems that are difficult to manage. Similarly, neglecting scalability considerations can result in systems that buckle under increasing loads, hindering performance and user experience. Another critical aspect is security, which should never be an afterthought; failing to integrate robust security measures from the outset can leave systems vulnerable to breaches and unauthorized access. Furthermore, ignoring the need for proper monitoring and logging mechanisms can impede troubleshooting efforts and leave issues undetected. It's also essential to avoid monolithic architectures that lack modularity, making maintenance and scaling arduous tasks. Additionally, overlooking compliance requirements, user experience, and adequate testing can all lead to subpar system performance and user satisfaction. By steering clear of these pitfalls and embracing best practices, such as prioritizing simplicity, scalability, security, and maintainability, system.

SYSTEM DESIGNASPECTS

Once the analysis stage is completed, the next stage is to determine in broad outline form how the problem might be solved. During system design, we are beginning to move from the logical to physical level.

System design involves architectural and detailed design of the system. Architectural design involves identifying software components, decomposing them into processing modules and conceptual data structures, and specifying the interconnections among components.

Detailed design is concerned with how to package processing modules and how to implement the processing algorithms, data structures and interconnections of standard algorithms, invention of new algorithms, and design of data representations and packaging of software products. Two kinds of approaches are available:

- Top-down approach
- Bottom-up approach

Design of Code

Since information systems projects are designed with space, time and cost saving in mind, coding methods in which conditions, words, ideas or control errors and speed the entire process. The purpose of the code is to facilitate the identification and retrieval of the information. A code is an ordered collection of symbols designed to provide unique identification of an entity or an attribute.

Design of Input

Design of input involves the following decisions

- Input data
- Input medium
- The way data should be arranged or coded
- Validation needed to detect every step to follow when error occurs

The input controls provide ways to ensure that only authorized users access the system guarantee the valid transactions, validate the data for accuracy and determine whether any necessary data has been omitted. The primary input medium chosen is display. Screens have been developed for input of data using HTML. The validations for all important inputs are taken care of through various events using JSP control.

Design of Output

Design of output involves the following decisions

- Information to present
- Output medium
- Output layout

Output of this system is given in easily understandable, user-friendly manner, Layout of the output is decided through the discussions with the different users.

Design of Control

The system should offer the means of detecting and handling errors.

Input controls provides ways per

- Valid transactions are only acceptable
- Validates the accuracy of data
- Ensures that all mandatory data have been captured

All entities to the system will be validated. And updating of tables is allowed for only valid entries. Means have been provided to correct, if any by change incorrect entries have been entered into the system they can be edited.

3.1 Database Design (E-R Diagram)

An Entity-Relationship (ER) model illustrates the structure of a database using a visual representation known as an Entity-Relationship Diagram (ER Diagram). This model serves as a blueprint for designing the database schema and capturing the relationships between different entities and attributes.

The ER model provides a systematic approach to organizing and conceptualizing the data within a database system. It represents entities as well as the relationships between them, helping to clarify how data elements are connected and organized.

Entity relationship diagram

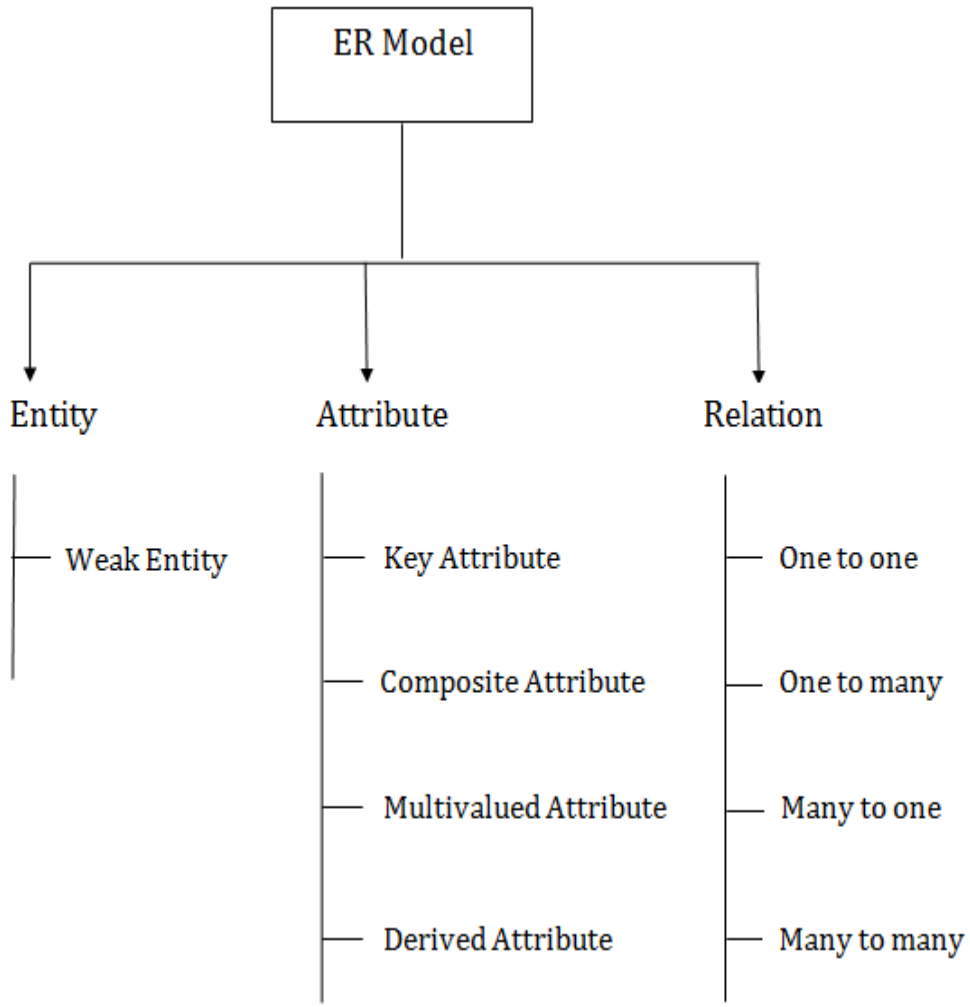
The Emergency Room model corresponds to an Entity-Relationship model, serving as a high-level representation of data structures. It is utilized to illustrate the data components and relationships within a defined system.

It establishes a structured framework for the database. Moreover, it provides a straightforward and easily understandable perspective on the data.

In Entity-Relationship modelling, the organizational database structure is depicted through a design known as an Entity-Relationship diagram.

For instance, consider designing a school database. An educational record could be represented as an entity with attributes such as name, ID, age, etc. Similarly, the address could be another entity with attributes like city, street name, zip code, etc., and there would be a relationship between them.

Component of ER Diagram



Entity

A substance may be anything, class, individual or spot. In the ER frame, a substance can be tended to as square shapes.

Weak Entity

A substance that depends upon another component called afrail substance. The frail element contains no critical trait of its own. The feeble substance is addressed by a twofold square shape

Attribute

The quality is utilized to depict the property of a section. Obscure is utilized to address a quality.

Key Attribute

The key quality is used to address the essential ascribes of a substance. It tends to a fundamental key. The key property is tended to by a circle with the text underlined**.**

Composite Attribute

A property that made from various attributes is known as a composite quality. The composite trademark is tended to by an oval, and those circles are related with a circle.

Multi valued Attribute

A quality can have more than one worth. These qualities are known as a multivalued property. The twofold oval is used to address multivalued property.

Derived Attribute

A property that can be gotten from another quality is known as a decided attribute. It will in general be tended to by a ran circle.

Relationship

A relationship is used to depict the connection between substances.
Important stone or rhombus is utilized to address the relationship.

Sorts of relationship are as per the following:

One-to-One relationship

At the point when just a single instance of a component is connected with the relationship, then it is known as facilitated relationship. For instance, A female can wed to one male, and a male can wed to one female.

One-to-many relationship

Exactly when simply a solitary illustration of the substance on the left, and more than one event of a component on the right associates with the relationship then this is known as a one-to-various connections.

For example, Scientist can envision various manifestations, but the improvement is done by the really express analyst.

Many-to-one relationship

Exactly when more than one event of the component on the left, and simply a solitary event of a substance on the right associates with the relationship then it is known as a many-to-one relationship.

For example, Student enrolls for only a solitary course, but a course can have various students.

Many-to-many relationship

At the point when more than one event of the substance on the left, and more than one event of a component on the right associates with the relationship then it is known as a many-to-various connections.

For example, Employee can allot by numerous exercises and project can have various special

ER diagram

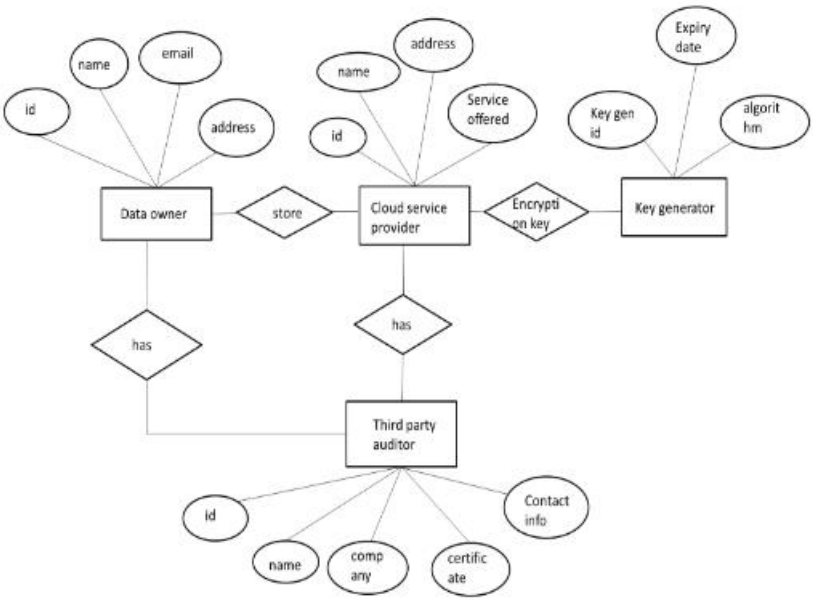


Fig. no 3.1.2E-R Diagram

3.2. Data Dictionary

A Data Dictionary compiles names, definitions, and attributes concerning data elements utilized or stored within a database, information system, or part of a research project. It delineates the meanings and functions of data elements within the context of a project and offers guidance on understanding, recognizing meanings, and description. Additionally, a Data Dictionary offers metadata about data elements, aiding in defining the scope and attributes of data elements, as well as the guidelines for their usage and application

Table Name: Data owner

Column name	**Data type**	**Size**	**constraints**
Id	Number	30	Primary key
Name	Varchar2	30	Not null
email	Varchar2	40	Not null
address	Varchar2	30	Not null

Table 3.2.1 Data Owner

Table Name: Cloud Service Provider

Column name	**Data type**	**Size**	**constraints**
Id	Number	30	Primary key
Name	Varchar2	30	Not null
address	Varchar2	30	Not null
Service offered	Varchar2	30	Not null

Table 3.2.2 Cloud Service Provider

Table Name: Key Generator

Column name	**Data type**	**Size**	**constraints**
Key generator Id	Number	30	Primary key
Expiry date	Date	-	Not null
algorithm	Integer	40	Not null

Table 3.2.3 Key Generator

Table Name: Third Party Auditor

Column name	Data type	Size	constraints
Id	Number	30	Primary key
Name	Varchar2	30	Not null
company	Char	40	Not null
certificate	Varchar2	30	Not null
Contact info	Number	20	Not null

Table 3.2.4 Third Party Auditor

3.3 UML Design

UML is an abbreviation for Unified Modelling Language. The UML has established itself as the software blueprint language of choice for analysts,designers,andprogrammersalike.TheUMLprovidesaconsistentvocabularyforeveryoneinvolveinsoftwaredesign,frombusinessanalysts to designers to programmers. The unified modelling languageenables a software engineer to represent an analysis model using modellingnotation, which is governed by a set of syntactic, semantic, and paradigmrules. A UML system is represented by five separate perspectives, each of which describes the system from a distinct point of view. Eachview isspecifiedbythefollowingsetofdiagrams.

UML is specifically constructed through two different domains they are:

- UML Analysis modelling, this focuses on the user model and structural model views of the system.
- UML design modelling, which focuses on the behaviouralmodelling, implementation modelling and environmental model views.

GOALS:

The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

Why We Use UML in projects?

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software and to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. Businesses also seek techniques to manage the complexity of systems as they increase in scope and scale. In particular, they recognize the need to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing and fault tolerance. Additionally, the development for the World Wide Web, while making some things simpler, has exacerbated these architectural problems. The Unified Modeling Language (UML) was designed to respond to these needs. Simply, Systems design refers to the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements which can be done easily through UML diagrams.

UserModelView

Thisviewrepresentsthe systemasseenbytheuser.Theanalyticaldepictiondepictsa usagescenarioasseenbythe end-user.

StructuralModelView

Thedataandfunctionalityinthismodel areobtainedfrom withinthe system. The static structures arerepresentedbythismodelview.

Behavioral Model View

Itrepresentstheinteractionsofcollectionbetweenvariousstructuralelementsde scribed intheusermodelandstructuralmodelview,representing the dynamicofbehaviouralassystemparts.

Implementation Model View

The structural and behavioural components of the system are depicted astheywillbebuiltinthismodel.

Environmental Model View

This represents the structural and behavioural aspects of the environment inwhichthesystemwillbedeployed.

Basic Building Blocks of UML

Objects are first-class citizens in a model; relationships connect them; anddiagrams group interesting collections of objects. The UML vocabularyincludesthreetypesofbuildingblocks.

- ✓ Things
- ✓ Relationships
- ✓ Diagrams

Things in UML

Thesearethefundamentalobject-orientedbuildingpiecesofUML.InUML, there arefourtypes ofthings.

- o StructuralThings.
- o BehaviouralThings.

- GroupingThings.
- An notationalThings.

Structural Things

The nouns of UML models are structural objects.These are the primarilystaticaspectsofamodelthatrepresenteither conceptual lorephysicalelements.There are sevendifferenttypesofstructuralthings.

Class

Aclassisadescriptionofagroupofobjectswithsimilarfeatures,operations, relationships, and semantics.A class implements one or moreinterfaces. A class is represented graphically as a rectangle, which often includes its name, attributes, and operates

Windows
Origin size
Open()C lose()Mo ve() Display()

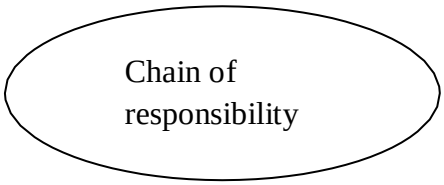
Interface

Aninterfaceisasetof operationsthatdefineaclassorcomponent'sservice.Asaresult,an interfaceexplainstheelement'sexternallyobservable behaviour. An interface may represent the entire behaviour of aclassorcomponent, or merely portionofit.Asetofoperatingspecificationsisdefinedbyaninterface.Aninterfa ceisrepresentedgraphicallyascirclewith its name.

Collaboration

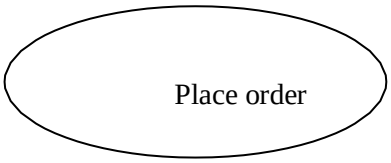
Collaboration is a society of roles and other aspects that work together togenerate some cooperativebehaviour thatismore than the sum of theelements.As a result, collaborations have both structural and behaviouraldimensions.A singleclassmaybeinvolvedinmanycollaborations.

Collaborationis represented visually a panellise with dashed lines,usually including merely its name.



Use Case

A use case is a description of a series of actions that a system takes that results in an observable result of value to a specific actor.A use case isusedto organize the behaviouralaspectsofamodel.



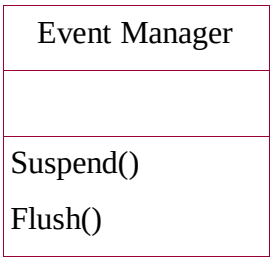
Active Class

Active classes, components, and nodes are all class-like, which means theydefineasetofobjectsthathavethesameproperties,operations,relationships , and semantics.However, these three are sufficiently distinct and required for modelling specific parts of an object-oriented system todeserve specialconsideration.

An active class is one whose objects possess one or more processes orthreads and so have the ability to initiate control activities.An active class

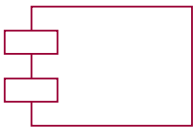
is similar to a class in that its objects represent elements whose behaviour is concurrent with the behaviour of other components.An active class

isrendered graphically in the same way as a class, but with heavier lines,usually featuringitsname,properties,and operations.



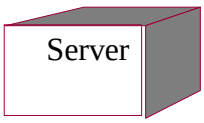
Component

A component is a physical and replacesableportionofasystemthatconforms to and implements a set of interfaces. There are various types ofdeploymentcomponents inasystem,suchasJavaBeans,aswellascomponents that are artefacts of the development process, such as sourcecodefiles.Acomponentisusuallythe physicalpackaging oflogicalpiecessuch as classes, interfaces, and collaborations. A component is representedgraphicallyasarectanglewithtabs,usuallycontaining onlyitsname.



Node

A nodeisaphysical elementthatexistsatrun timeandrepresentsacomputing resource, typically with some memory and, more often than not, processing capabilities. A collection of components may reside on a node and may migrate from node to node. A node is represented graphically as acute, withsimplyitsnamedisplayed.



Classes, interfaces, collaborations, use cases, active classes, components,and nodes are the seven main structural elements that can be included in aUML model.Variations on these seven include actors, signals, and utilities(class types), processes and threads (active class types), and applications,documents,files,libraries,pages,and tables.

Behavioral Things

Behaviours are dynamic components of UML models.These are the verbsof a model that depicts behaviour across time and space. Ingeneral, therearedtwotypes ofbehaviouralthings.

To begin, an interaction is a behaviour that consists of a set of messagessent among a set of objects in a certain context to accomplish a specificgoal. An interaction might specify the behaviour of an object society or the behaviour of an individual operation. Other components of an

interactionincludemessagesandactionsequences.Messagesaregraphicallyre presented as directed lines that almost always include the name of the operation.

Second, a state machine is a behaviour that specifies the sequence of statesan object or an interaction goes through during its lifetime in response to events, as well as its responses to those events. A state machine can specify the behaviourofan individualclassora collaborationofclasses.

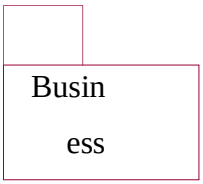
A state machine also includes states, transitions (the flow from one state toanother), events, and activities (the response to a transition).A state istypically represented graphically as a rounded rectangle that includes itsname andsubstates.

Interactions and state machines are the basic behaviour aspects that can beincludedin aUMLmodel.Theseelementsare typically semantically linked to various structural elements, including classes, collaborations, and objects.

Grouping Things

The original components of UML models are grouping things. Theseare the boxes that can be used to breakdown a model. Overall, there is just one type ofgroupingthing,andthatispackages.

A package is a generic approach for grouping things into k8groups.Apackage can contain structural items, behavioural items, and even other groupingitems. A package, unlike components (which exist at run time), is solely conceptual. A package is represented graphically as a tabbed folder,generallycontainingmerelyitsnameand,occasionally,itscontents.



Annotational Things

Notational elements are the explanatory components of aUMLmodel.Thesearethecommentsthatyoucanusetodescribe,enlighten,and comment on any element in a model. A note is the most basic type ofannotational object. A note is merely a symbol for attaching rendering limitations and comments to an element or group of elements. Agraphic note is shown as erectanglewith ado-eared cornered textual orgraphicalstatement.

This is the only fundamental an notational element that can be included in alummodel. Notes are often used to embellish diagrams with constraintsorcommentsthatare beststatedintext,whethercasualorformal.

Relationships in UML

These are the fundamental relational building pieces of the UML.They areusedtocreatewell-formedmodels.InUML,therearefourtypesofrelationships.

- Dependency
- Association
- Generalization
- Realization

Dependency

To begin, dependence is a semantic relationship between two objects in which a change to one of them (the independent thing) might impact thesemanticsoftheother(thedependentthing).Dependencyisshowngraphicall y as a dashed line, possibly directed, and occasionally with label.

Association

Second, an association is a structural relationship that describes a collection of links, each of which is a connection between things. Aggregation is type of association that represents a structural relationship between a whole and its constituent pieces. An association is shown graphically as a solid line, possibly directed, occasionally with a label, and frequently with extraadornmentssuchasmultiplicityandrolenames.

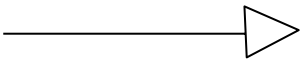
Employer **Employee**

Generalization

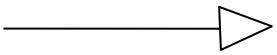
Finally,ageneralizationisspecialization/generalization connectioninwhich objects of the specialized element(the child) are interchangeable with objects of thegeneralizedelement(the parent).In this way,theyoungsteradoptstheparent'sstructureandconduct.Ageneralizationrelationshipisrepresentedgraphicallyasasolidlinewithahollowarrowheadpointingtotheparent.

Realization

Fourth, a realization is a semantic link between two classifiers in which



oneclassifierspecifiesacontractthattheotherclassifierguaranteestofulfill.Realization relationships can be found between interfaces and the classes or



Componentsthatrealizethem,aswellasbetweenusecasesandthecollaborationsthatrealizethem.

Arealizationrelationshipisshowngraphically as a cross between a generalization and a dependency relationship.

These four elements are the fundamental relational elements that can be included in a UML model. Variations of these four include refinement,trace,include,andextended(for dependencies).

UMLdiagramsaregraphicalrepresentationsofacollectionofitemsthatare oftenrenderedasaconnectedgraph of vertices(objects)and arcs(relationships).Diagrams are used tovisualizea system fromvariousperspectives, so a diagram provides an omitted view of the elements that comprise a system. The same element may occur in all diagrams, a subset ofdiagrams,ornoneatall.Adiagramcantheoreticallycontainanycombinationofthingsand relationships.

DiagramsinUML

A diagram is a graphical representation of a collection of items, which istypically represented as a connected network of vertices (objects) and arcs(relationships).Diagramsareusedtovisualizeasystemfromvariousperspe ctives, so a diagram provides an omitted view of the elements that comprise a system. The same element may occur in all diagrams, a subsetofdiagrams,ornoneatall.Adiagramcantheoreticallycontainanycombin ationofthings and relationships.

However, inpractice, smallnumber of common combinationsemergethat are consistent with the five most relevant viewpoints that compose the architecture of a software intensive system. As a result, the UML includes ninesuch diagrams.

- UseCaseDiagram
- Class Diagram
- Object Diagram
- Sequence Diagram
- Collaboration Diagram
- Component Diagram
- Deployment Diagram
- StatechartDiagram
- Activity Diagram

Use Case Diagram

Use Case diagrams are one of five diagrams in the UML for modelling the dynamic features of systems. The other four diagrams in the UML for modelling the dynamic aspects of systems are activity diagrams, sequence diagrams, status chart diagrams, and collaboration diagrams. UseCase diagrams are essential for modelling the behaviour of a system, subsystem, or class. Each one depicts a certain collection of use cases, actors, and interactions.

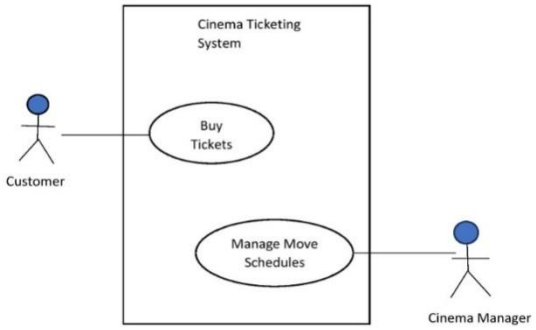


Fig3.3.1: Use Case Diagram

Class Diagram

Class diagrams are the most commonly used diagrams in the modelling of object-oriented systems. A class diagram depicts the links between classes, interfaces, and collaborations. A class diagram is a graphical representationofvertices and arcs.

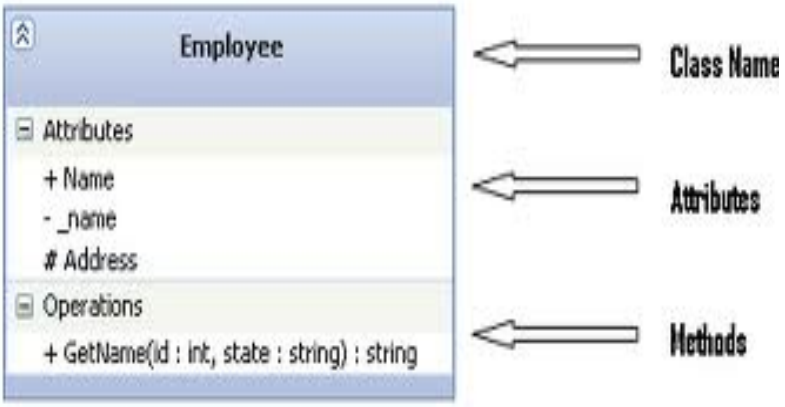


Fig 3.3.2: Class Diagram

Collaboration Diagram

An interaction diagram depicts an interaction, which consists of a set of objects and their relationships, as well as the messages that can be sent.

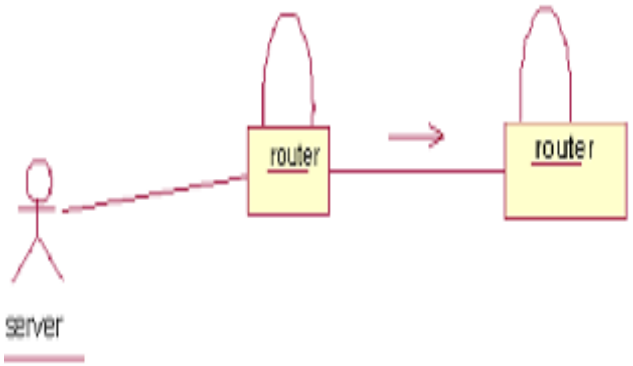


Fig3.3.3: Collaboration Diagram

Sequence Diagram

A sequence diagram is an interaction diagram that emphasizes messagetimeordering.Asequencediagramisagraphicalrepresentationofatable with items grouped along the X-axis and messages sorted in increasing timealong they-axis.

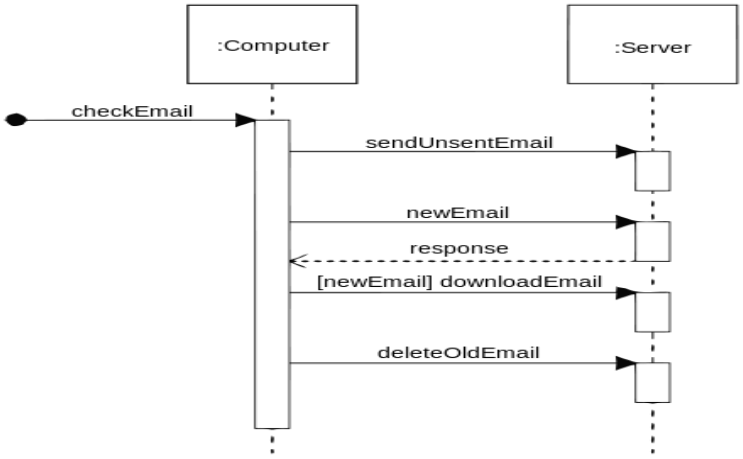


Fig.3.3.4: Sequence Diagram

Activity Diagram

An Activity Diagram is essentially a flow chart that depicts the flow ofcontrolfrom oneactivitytothenext.Theyareusedtosimulatethesystem's dynamic features. They can also be used to simulate the flow of an object as it transitions from one state to another at various points in thecontrolflow.

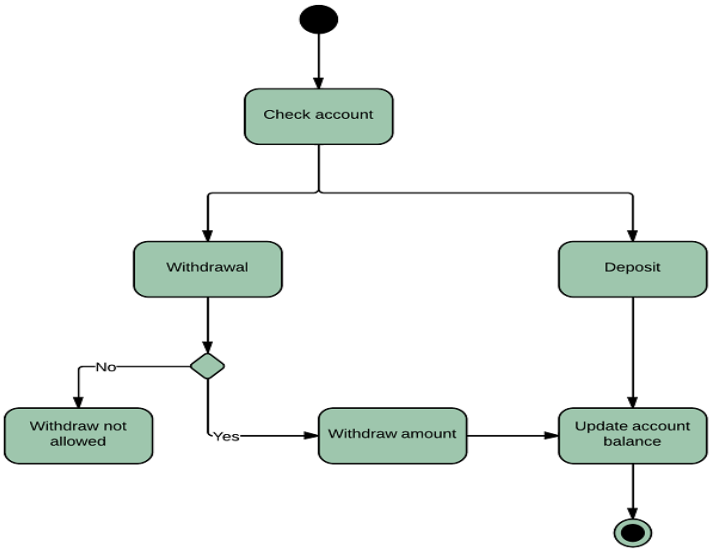


Fig3.3.5: Activity Diagram

State Chart Diagram

A state machine appears in a state chart diagram. State chart diagrams aroused to model the system's dynamic features. The majority of the time, this entails simulating the behaviour of the reactive objects. A reactive object is one whose behaviour is best defined by its reaction to events that occur outside of its context.

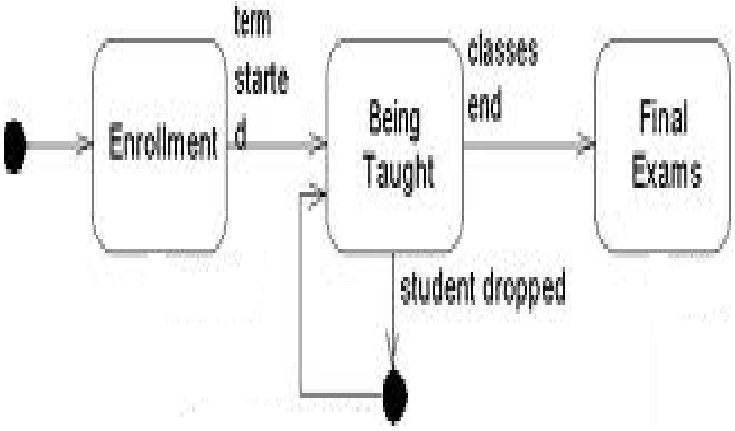


Fig3.3.6: State Chart Diagram

Component Diagram

The component diagrams for the selected component package, as well asnewentriesinthislist,alwaysincludeanentryforthecomponentpackage'smai ncomponentdiagram.

Fig 3.3.7: Component Diagram

Deployment Diagram

Processors, devices, and connections are depicted in a deployment diagram. Each model has a single deployment diagram that depicts the relationships between its processors and devices, as well as the process allocation to processors.

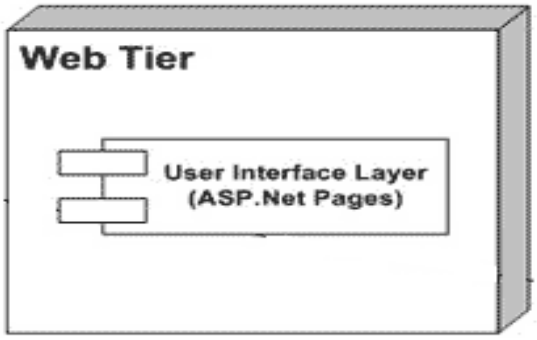


Fig 3.3.8: Deployment Diagram

USE CASE DIAGRAM

The use cases within this cloud computing project encompass essential functionalities for secure data management and verification. Firstly, theDeduplicate Data use case involves cloud users initiating requests to the

Cloud Service Provider (CSP) to optimize storage space and bandwidth by identifying and removing redundant data instances.

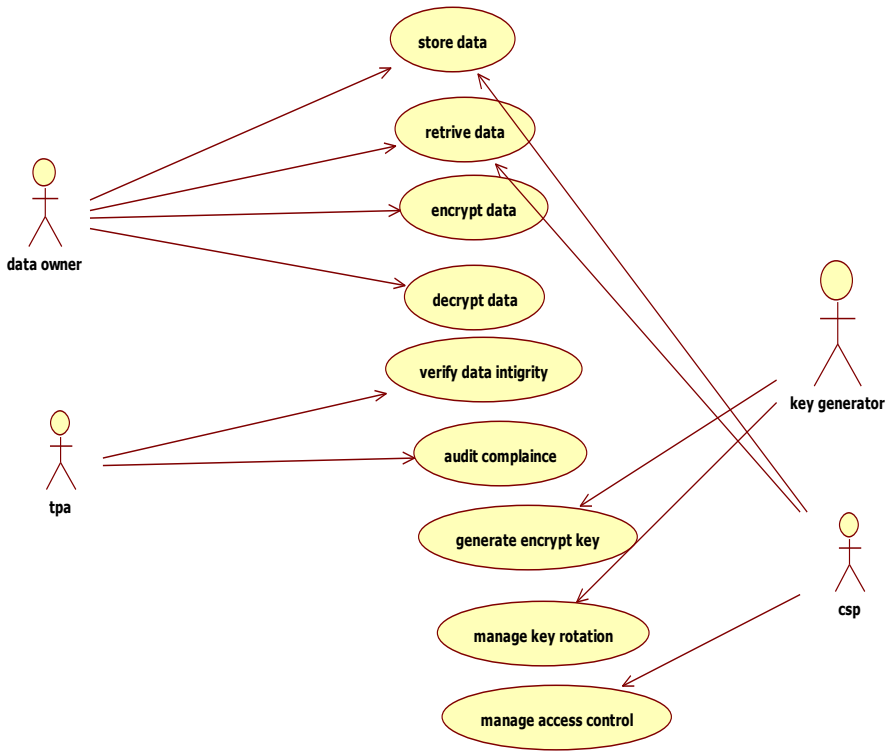


Fig 3.3.9: Use case Diagram for overall project

Description: The figure describes about Use Case diagram of Audio to Sign Language Translator process. A use case is a description of a set of sequences of actions, including variants that a system performs to yield an observable result of value to an actor. Graphically, a use case is rendered as an ellipse.

CLASS DIAGRAM

A class diagram is a type of UML (Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes in a system. It provides a high-level view of the system's architecture by showing the classes, their attributes, methods, and the associations between them.

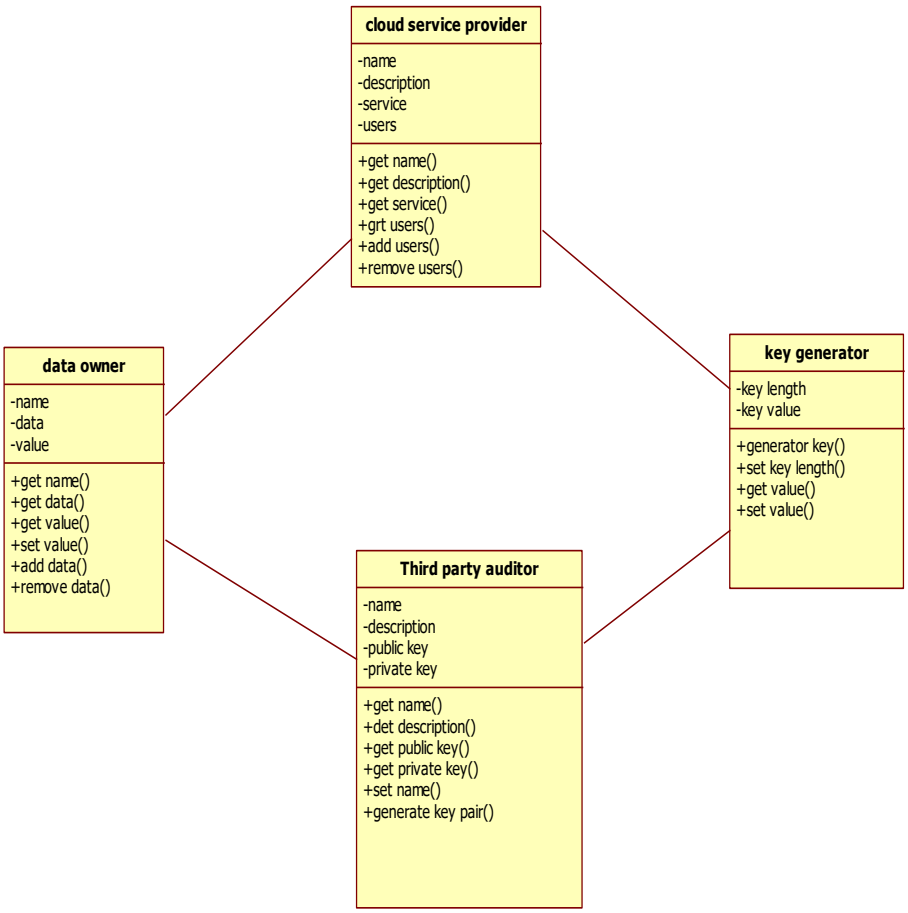


Fig3.3.10: Class Diagram for Overall Project

Description: A class diagram serves as a visual blueprint for software systems. These connections include associations, which show how classes relate, inheritance, indicating when one class inherits characteristics from another, and dependencies, signifying reliance between classes.

SEQUENCE DIAGRAM

A Sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

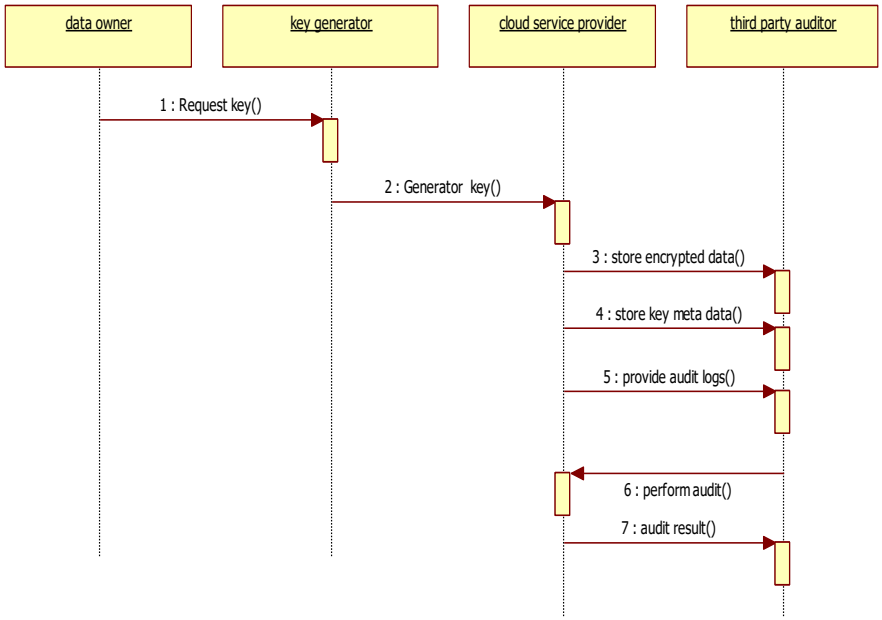


Fig 3.3.11: Sequence Diagram for Overall Project

Description: The above figure represents the sequence diagram of Audio to Sign Language Translator process. Which describes sequence of actions or tasks takes place in this project.

ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

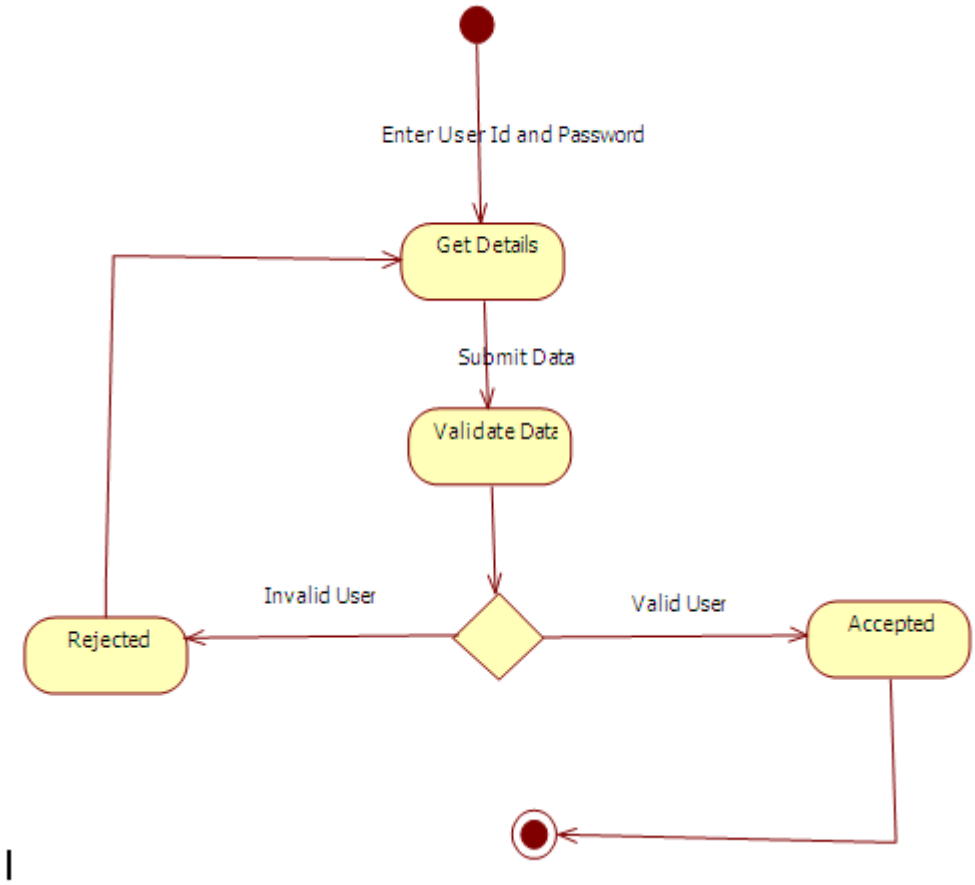


Fig 3.3.12: Activity Diagram

Description: The above diagram shows the activity diagram of Audio to Sign Language Translator process. An activity diagram shows the flow from activity to activity. It is ongoing non atomic execution within a state machine.

DEPLOYMENT DIAGRAM

Deployment diagrams describe the configuration of run-time processing resource elements and the mapping of software implementation components on to them. These diagrams contain components and nodes, which represent processing or computational resources, including computers, printers etc.

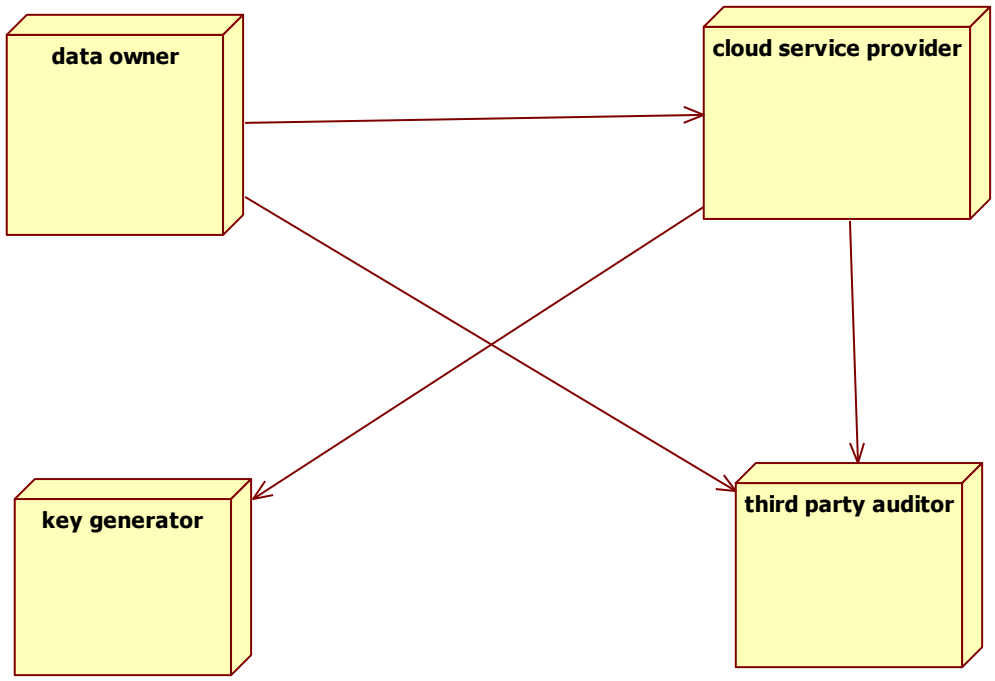


Fig 3.3.13: Deployment Diagram

Description: Deployment diagram is a collection of vartices and arcs. Common uses of deployment diagrams are modeling the embedded systems, to model the client/server system and to model the distributed systems.