**ONLINE COLLEGE BUS TRACKING SYSTEM**

**A Mini Project Report submitted under the**

OBJECT –ORIENTED SOFTWARE ENGINEERING LAB



Prepared by

CH. Pavan Kumar 160105032

**Under the Esteemed guidance of**

Dr. S. VENKATARAMANA

**Department of Information Technology**

**S.R.K.R.EngineeringCollege**

China -Amiram, BHIMAVARAM-534204

**ABSTRACT**

The modern world is guided by the change in the technology day by day. Mostly the relevant changes in technologies are enhancing the modern business techniques. Different technologies have been developed in the world for making people’s life easier and better day by day. Android is the latest and a rapid growing technology available for all the users or users in today’s market. An enormous increase in the end user acceptance has been experienced in the past few years. The project is based on the latest GPS technology which enables college management team a better way to keep eye on the activity of the college buses and manage schedule as well as provide real time bus location for the students using bus service. This paper proposes an Android mobile phone application that gives information about buses, bus numbers as well as bus routes/stops online. The proposed system is completely integrated online bus tracking systems using database. It provides the facility of tracking the particular college bus’s location in the google map. They can also view the bus details such as bus schedule and they reach the bus on time.

INDEX

**1. INTRODUCTION**

1.1 SCOPE

1.2 UML

1.3 BASIC BLOCKS OF UML

1.4 SOFTWARE ENGINEERING ACTIVITIES

1.5 RATIONAL ROSE

**2. PROBLEM SPECIFICATION**

2.1 EXISTING SYSTEM

2.2 PROPOSED SYSTEM

**3. SYSTEM REQUIREMENTS**

3.1 SYSTEM REQUIREMENTS

3.2 FUNCTIONAL REQUIREMENT

**4. USER INTERFACE DESIGN**

4.1 SAMPLE USER INTERFACE SCREENS

**5. ANALYSIS**

5.1 INTRODUCTION TO ANALYSIS

5.2 USE CASE MODEL

5.2.1 INTRODUCTION TO USE CASE MODEL

5.2.2 USE CASE DIAGRAM

5.3 INTERACTIONAL DIAGRAM

5.3.1 INTRODUCTION TO INTERACTION DIAGRAM

5.3.2 SEQUENCE DIAGRAM

5.3.3 COLLABORATION DIAGRAM

5.4 STATECHART DIAGRAM

5.4.1 INTRODUCTION TO STATECHART DIAGRAM

5.4.2 STATECHART DIAGRAM

5.5 CLASS DIAGRAM

5.5.1 INTRODUCTION TO CLASS DIAGRAM

5.5.2 CLASS DIAGRAM

**6. SYSTEM DESIGN**

6.1 INTRODUCTION TO SYSTEM DESIGN

6.2 COMPONENT DIAGRAM

6.3 DEPLOYMENT DIAGRAM

**7. IMPLEMENTATION**

7.1 INTRODUCTION TO IMPLEMENTATION

7.2 INTRODUCTION TO ACTIVITY DIAGRAM

7.3 ACTIVITY DIAGRAM

**8. TESTING**

8.1 INTRODUCTION TO TESTING

8.2 SAMPLE TEST CASES

**9. ER-DIAGRAM AND DATABASE DESIGN**

**10. CONCLUSION**

**11. REFERENCES**

**INTRODUCTION**

**College Bus Tracking System** is a system developed on Android Platform using java programming language. It is based on client-server technology along with the use of database. One Android user (College Bus Driver) sends real time location of the bus with additional date and time information to the server. The information provided by that user is stored in the database of the server. And other android users can get the information through the server. The login page is available on the user app for the college administrator. The administrator can keep the record of the bus such as bus no., bus schedule, route info, driver contact, etc. on the database. The administrator also has the permission to manipulate the bus record as per the needs. Student need to login. Student can search for the particular bus on the map. Students get updated on the bus location at certain time interval so that they don’t have to wait for the bus being unknown whether the bus is coming or has gone. So in summary, our system handles all the data about current location of bus and by using this data the real time tracking of bus can be done and this information is then given to remote user who want to know the real time bus information. For development purpose some technologies like GPS (Global Positioning System) and Google maps are used. The system includes server-client based application, which gives real time location of bus on Google Maps.



Fig1: GPS Technology

1.1**.Scope:**

* Any colleges/schools/travelling agencies  can make use of this application providing information about the live location of the vehicles.
* This system would provide basic set of features such as adding or updating routes and locations in the database.

**1.2 UML:**

* Unified Modelling Language is the modelling language used for representing the models. The goal of UML is to provide a standard notation that can be used by all object oriented methods to select and to integrate the best elements of precursor notations. System development focuses on three different models.
* Functional model: It describes the functionality of the system in user's point of view. The functionality will be expressed in terms of use cases and actors. This is nothing but the use case diagram.
* Object model: This model describes the system in terms of classes, objects, attributes, operations and relation among classes. The object model starts with analysis object model in analysis phase and it is transformed into system design object model in system design phase and finally refined into object model in object design phase. This model is represented with class diagrams.
* Dynamic model: This describes the internal behaviour of the system. This model is represented with interaction diagrams, state chart diagrams and activity diagrams.

**1.3 Basic Blocks of UML:**

The Basic building blocks of the UML are:

* Things
* Relationships
* Diagrams

**Things:**

Things are the most important building blocks of UML. Things can be like

* Structural
* Behavioral
* Grouping
* Annotational

**i. Structural Things:**

Structural things define the static part of the model. They represent the physical and conceptual elements. Following are the brief descriptions of the structural things.

**Class −** Class represents a set of objects having similar responsibilities.

class

**Interface −** Interface defines a set of operations, which specify the responsibility of a class.

Interface

**Collaboration −**Collaboration defines an interaction between elements.

Collaboration

**Use case −**Use case represents a set of actions performed by a system for a specific goal.

Use case

**Component −**Component describes the physical part of a system.

Component

**Node −** A node can be defined as a physical element that exists at run time.



**ii. Behavioral Things:**

Abehavioral thing consists of the dynamic parts of UML models. Following are the behavioral things −

**Interaction −** Interaction is defined as a behaviour that consists of a group of messages exchanged among elements to accomplish a specific task.

Interaction

**State machine −** State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change



**Grouping Things:**

Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available −

**Package −** Package is the only one grouping thing available for gathering structural and behavioural things.



**iii. Annotational Things:**

Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements.

Note

**Relationship:**

Relationship is another most important building block of UML. It shows how the elements are associated with each other and this association describes the functionality of an application.

There are four kinds of relationships available.

**Dependency:**

Dependency is a relationship between two things in which change in one element also affects the other.

Dependency

**Association:**

Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.

Association

**Generalization:**

Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world of objects.

Generalization

**Realization:**

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.

Realization

**UML Diagrams:**

UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system.

The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it complete.

UML includes the following nine diagrams, the details of which are described in the subsequent chapters.

* Class diagram
* Object diagram
* Use case diagram
* Sequence diagram
* Collaboration diagram
* Activity diagram
* State chart diagram
* Deployment diagram
* Component diagram

**1.4 Software Engineering Activities:**

Software engineering activities deal with the complexity by constructing and validating models of the application domain model or the system. Engineering activities include

* Requirement Elicitation
* Analysis
* System Design
* Object Design
* Implementation
* Testing

**Requirements Elicitation:**

During requirements elicitation, the client and developers define the purpose of the system. The result of this activity is a description of the system in terms of actors and use cases. Actors are those who interact with the system. Actors may be end users or the computers that interact with the system. Use case diagram will be drawn here.

**Analysis:**

During analysis, d elopers aim to produce the model of the system that is collect, complete, consistent and unambiguous. The system will be modified from use cases into object model that completely describes the system. Class diagram, Object diagram, Interaction diagrams such as sequence diagram and collaboration diagram, State chart diagram will be drawn in this phase.

**System Design:**

During the system design goals of the project and decompose the system into smaller subsystems that can be realized by the individual teams. This also involves strategy selection which includes access control policy, data flow policy, data management, hardware configuration, software configuration. Component diagram will be drawn here.

**Object Design:**

During object design, developers define solution domain objects to bridge the gap between the analysis model and the hardware/software platform defined during the system design. The result of the object design activity is a detailed object model annotated with constraints and precise descriptions for each element.

**Implementation:**

During implementation, developers translate the solution domain model into source code. This includes implementing the attributes and methods of each object integrating all the objects such that they function as a single system.

**Testing:**

During testing, developers find differences between the system and its models by executing the system with sample input data sets. During unit testing, developers compare the object design model with each object and subsystem. During integration testing, combinations of subsystems are integrated together and compared with the system design model. During system testing, typical and exception cases are run through the system and compared with the requirements model. The goal of testing is to discover as many faults as possible such that they can be repaired before the delivery of the system.

**1.5 Rational Rose:**

The Rational Rose section of the rose.ini file contains startup settings for the application. The application reads and writes to this file upon opening and closing the application. Most settings in the rational rose will be altered when you change the options through the GUI. The only way you can change some settings is directly in the rose.ini file. Rational Rose automatically allows you to integrate other applications with Rational Rose in two ways.

Using Rational Rose as an automatic controller, you can call an OLE automation object from within a Rational Rose script. Using Rational Rose software as an automatic server, you can call its OLE-compliant applications.

Rational Rose automatic is accessible to automatic controller environments such as visual basic, EXCEL, Summit Basic Script, C, C++ and others.

**2. PROBLEM SPECIFICATION**

**2.1 EXISTING SYSTEM:**

* Location of the bus is not known, thus we need to ask the bus information for respective persons.
* Long time waiting for the bus.
* Chances of missing bus.

## 2.2 PROPOSED SYSTEM:

* We are expecting to find the location of the bus and let the users know the location.
* so that one can manage their time efficiently and reach their stop just before the bus arrives or take an alternate means of transport if they miss the bus or they are running late.

**3. SYSTEM REQUIREMENTS**

**SOFTWARE REQUIREMENTS:**

* Technology used : Android 4.1 or higher
* IDE : Android Studio
* Emulator : Android Emulator
* Plug-in : ADT plug-in
* Back-End : php, SQLite
* Front-End : Android SDK

**HARDWARE REQUIREMENTS:**

* System Processor : Microcontroller
* Mobile Processor : 1GHZ or higher
* Other Requirements: 12v power supply, GPS Receiver, GSMModule

3.2FUNCTIONAL REQUIREMENTS:

These are the functions done by the system such as:

* Store the student information in database
* Store the bus information in database
* Update location in database
* Retrieve the results from database

This system generates continuous update of location, where the students can

see the location before starting and reach their stop on time with no delay.

**Store the student information in database:**

The students who has registered for bus has to get logins from the college and also bus driver should register and get the login from management.

This information is stored in database.

**Store the bus information in database:**

Store the information of bus such as bus id, starting point of the bus.

**Update location in Database:**

The Driver first connect the device to network and initiates the update location function while starting the bus and location details sent to database and updated for every minute.

**Retrieve results from Database:**

When the students click track button for the corresponding bus, the location of the bus is fetched from database and displayed to student by using user interface screen.

**NON FUNCTIONAL REQUIREMENTS:**

**Usability:**

Any student familiar with android applications can use this application since it has user friendly interface.

**Reliability:**

Since we using GPS tracker combined with GSM module to get the location so, the location we are getting is correct and accurate and location details are controlled by the admin.

**Performance**:

The online bus tracking system can operate its function in less than a second to get the location and to update the location in the database

and the student can view the location of the bus concurrently.

**Operation**:

The location and bus details are controlled by the college administrator.

**Supportability**:

As it is a android application it can be installed in only android OS

and developer maintains this application.

**4. USER INTERFACE DESIGN**

**4.1 SAMPLE USER INTERFACE SCREENS**

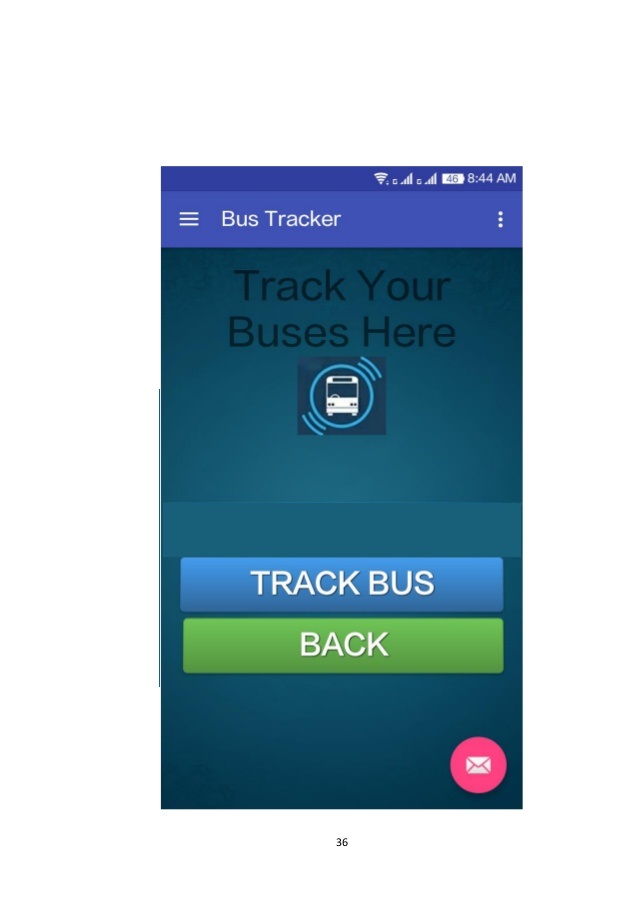


Fig: Track Bus page

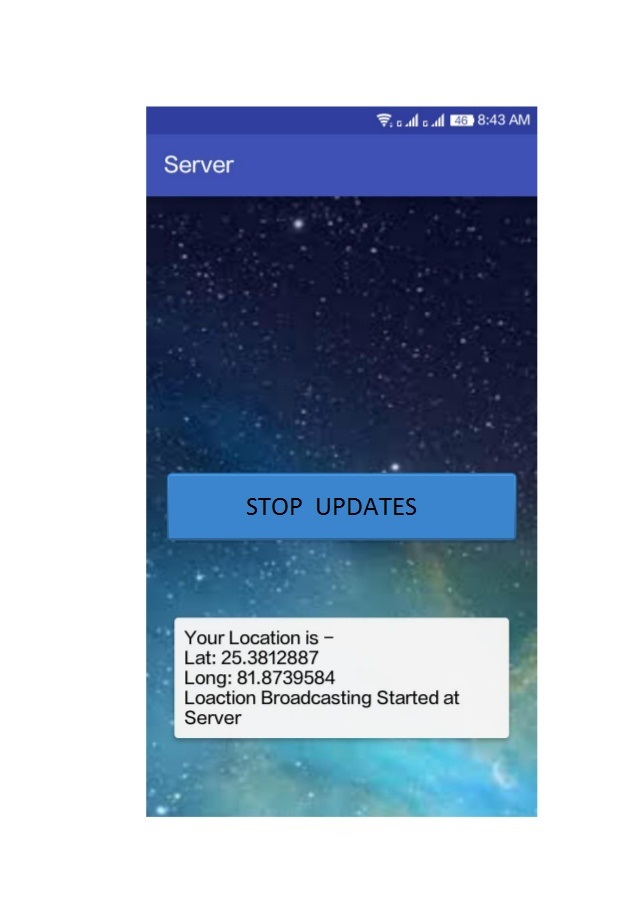
****

Fig: Stop updates screen

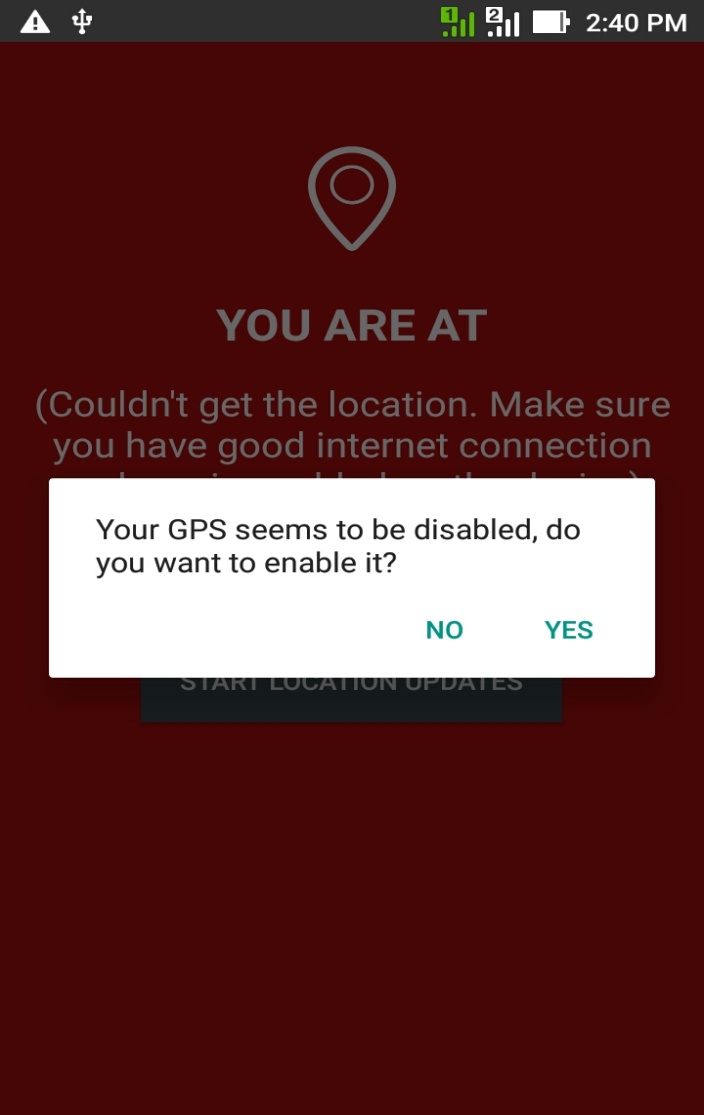


Fig: Prompt for setting

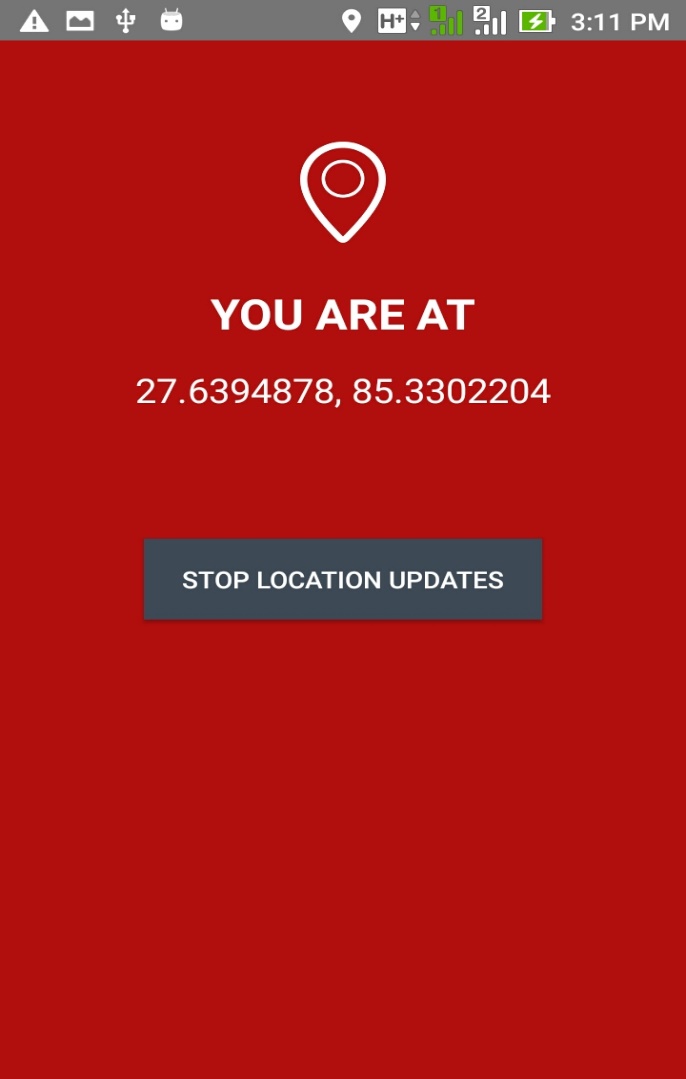


Fig: Show current location with updates

**5. Analysis**

**5.1 Introduction to Analysis:**

During Analysis, developers aim to produce the model of the system that is correct, complete, consistent and unambiguous. The system will be modified from use cases into object model that completely describes the system. Class diagram, Object diagram, interaction diagrams such as sequence diagrams and collaboration diagram, State chart diagram will be drawn in this phase.

**5.2 Use Case Model:**

**5.2.1 Introduction:**

A use case diagram shows a set of use cases and actors and their relationships. Use case diagrams address the static use case view of system. These diagram are especially important in organizing and modelling the behaviours of the system.

**5.2.2 Use Case Diagram:**

A use case diagram commonly contains

* Use cases
* Actors
* Dependency, generalization and association relationships.

A Use case diagram may also contain packages, which are used to group elements of your model into larger chunks.

Use case diagram is used in one of two ways:

1. To model the context of a system.
2. To model the requirements of a system.

**Modelling Techniques:**

1. Identify the actors that surround the system by considering which groups require help from the system to perform their tasks; which groups are needed to execute the systems functions, which groups interact with external hardware or other software systems; and which groups perform secondary functions for administration and maintenance.
2. Organize actors that are similar to one another in a generalization/ specialization hierarchy where it aids understandability, provide a stereotype for each such actor.
3. Populate a use case diagram with these actors and specify the paths of communication from each other to the systems use cases.



Fig: Main use case

**Check Available Buses sub-use case:**

Name: Check available buses sub-use case

Participating actors: Student, Server

Entry condition: login to application

Flow of Events:

1.Click the check available buses button.  
 2. Display the buses which will pass through the student

location.

3. If buses available then track otherwise click alternate bus

button.

Exit Condition: Track



Fig : Check available buses sub usecase

**Login Sub-Use Case Diagram:**

Name: Login Sub-Use Case

Participated actors: Student/ Server

Entry Condition: Student should have account.

Flow of Events:

1.Register your Account.

2. Login using the credentials used while registering.

Exit Condition: Logout



Fig :Login Subusecase

**Track sub-use case Diagram:**

Name: Track sub-use case

Participating Actors: Students, Server

Entry Condition: Student login to application

Flow of Events:

1.Student click track button if bus is available.

2. He/She will get the bus location from database.

3.After entering the bus student will stop the updates.

Exit Condition: Logout.



Fig: Track Bus Sub usecase

**Update location sub-use case Diagram:**

Name: Update location sub-use case

Participating Actors: Driver , GPS System, Server

Entry Condition: Driver login to system

Flow of Events:

1. Driver connect the system to network.

2. Driver will enable the GPS.

3. He will click the update location button.

4. The location will be updated for every one minute in database.

Exit Condition: Logout.



Fig :Update Location Sub-use case

**5.3 Interaction Diagrams:**

Both sequence diagrams and collaboration diagrams are kinds of interaction diagrams.

**5.3.1 Sequence Diagrams:**

It displays interaction between objects that focus on the message from a temporal stand point. The sequence diagram representation focuses on expressing interaction. An object is represented by a rectangle and its life line is represented by a vertical bar and dashed line. Sequence diagram show interaction between objects in a system and also specify the sequence in which those interactions happen and add the dimension & in of time to your diagram. In sequence diagram we only talk about time and ordering but not about the duration of time. In sequence



Fig: available buses sequence diagram



Fig: track sequence diagram



Fig: Update location sequence diagram

**5.3.2** **Collaboration Diagrams:**

It represents collaboration diagram objects. It is nothing but a set of objects related in a particular content and interaction. In diagram numbering the message indicates the sequence. Collaboration diagram shows exactly the same information the sequence diagram shows. But the purpose of collaboration diagram is different. The collaboration diagram shows the actor interaction without reference to time. Quality assurance engineers and system architects view at these diagrams to see the distributions of processing between objects. A system architect then concludes that the system is too dependent on central objects and redesigns the objects to distribute the processing evenly. We get the collaboration diagram by pressing F5 after sequence diagram is drawn.  


Fig : Available buses collaboration diagram



Fig: Track collaboration diagram



Fig: Update location collaboration diagram

**5.4 StateChart Diagram:**

**5.4.1 Introduction:**

A state chart diagram shows state machines, emphasizing the flow of control from state to state. A state machine is a behaviour that specifies the sequence of state objects goes through during its lifetime in response to events, together with its response to that response, to those events. A state is a condition or situation in the life of an object during which it satisfies some conditions, performs some activity or waits for some event.

**5.4.2 State Chart Diagrams:**

State chart diagram commonly contain:

-simple states and composite slates.

-Transitions, including events and actions.

State chart diagram is used only one way.

-To model reactive objects.

Common Modelling Techniques:

1. Choose the content for the state machine, whether it is a class, a use case, or the system as a whole.

2. Choose the initial and final states for the object. To guide the rest of your model, possibly state the pre and post conditions of the initial and final states respectively.

3. Decide on the stable states of the object by considering the conditions in which the object may exist for some identifiable period of time.

4. Decide on the meaningful partial ordering of stable states over the lifetime of the object.

5. Decide on the events that may trigger a transaction.

6. Attach actions to these transitions and or to those states.

7. Consider ways to simplify your machine by using sub states, branches, forks, joins, and history states.

8. Check that all states are reachable under some combination of events.

9. Check that no state is a dead end from which no combination of events will transition the object out of that state.



Fig: Available buses state chart



Fig: Login state chart



Fig: Track state chart



Fig: Update location state chart

**5.5 Class Diagram:**

**5.5.1 Introduction:**

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. The diagrams are most common diagrams founding modelling object-oriented systems. Class diagram represents class name, attribute and method. It shares the same attributes, operations, relations and semantics.

Class – it is description of set of objects.

Attribute – it represents the property of a class.

Methods – performing operations on date.

A class is something that provides blueprint for all object.

In other words class defines what information and object can hold and the operations that can be performed on the attributes of the objects of class.

**Class Diagram Representation:**

|  |
| --- |
| Class Name |
| Attributes |
| Operations |

A class is a set of objects that share a common structure and common behavior. A class is an abstraction of real world items.

When these items exist in the real world, they are instances of the classes and are referred to as objects.

**Multiplicity:**

The cardinality field specifies the number of expected instances of the class. You can set a specific cardinality value for the client class, supplier class, or both. In the case of relationships, this field indicates the number of links between each instance of the client class and the instances of the supplier class.

Use the following syntax to express Cardinality:

Value                                                  Description

N (default)                                          Unlimited number of instances

1                                                           One instance only

0...n                                                      Zero or more instances

1...n                                                      one or more instances

0...1                                                      Zero or one instance

<Literal>\*                                            each number of instances

<Literal>…N         Exact number of instances

<Literal>…<Literal>                            Specified range of instances

<Literal>…<Literal>, <Literal>            the number of instances will be in the specified range or an exact number of instances

<Literal>…<Literal>, the number of instances will be in one of the specified ranges.

Where <Literal> is any integer greater than or equal to one. You can set or display cardinality through the specification.



Fig: College Bus Tracking Class Diagram

**6. System Design**

**6.1 Introduction:**

System design consists of

1. Design goals:

The design goals can be identified by the following criteria

* Performance criteria such as response time, throughput and memory.
* Dependability Criteria

1. Reliability

2. Security

3. Robust

4. Availability

* End User Criteria

This criteria can be done in three levels

1. by Giving Commands

2. by Giving Inputs

3. By using more user friendly  atmosphere

* Cost Criteria

This is the total cost that is spend on the resources including man power, time and everything used in the achievement of design goals of the project

* Maintenance Criteria

The first three criteria are related with the client whereas the remaining are related to those who are developing the Project.

2. Decomposing the system into many sub systems.

Each use case is a sub system and the sub system will be encapsulated by an interface. The coding for this interface will be present in facade design pattern.

3. Strategies

Identifying hardware and software configuration used for the project.

Access policy

Global control flow

Boundary conditions

**6.2 Component Diagram:**

Component diagram show a physical view of your model. A component diagram shows you the components in your system and the relationships between them. This high level physical component may or may not be equivalent to the many smaller components you use in creation of application. Component diagram provide a physical view of the current model. A component diagram shows the organizations and dependencies among software components, including source code components, binary code components, and executable components. These diagrams also show the externally visible behaviour of the components by displaying the interfaces of the components.

Component Diagram contain:

1. Component Packages
2. Components
3. Interfaces
4. Dependency Relationship

**Component Packages:**

Component packages represent clusters of logically related components, or major pieces of your system. Component packages parallel the role played by logical packages for class diagrams. They allow you to participate the physical model of the system.

**Naming:**

Typically, a component package name is the name of a file system directory.

**Relationships:**

A component package can have dependencies with other component packages, components, and interfaces.

**Graphical Depiction:**

The component package is a folder shape icon.



**6.3 Deployment Diagram:**

A deployment diagram shows the configuration of run-time processing nodes and the components that live on them.

Deployment diagrams address the static deployment view of architecture. A node typically hosts one or more artifacts. We can use import and generalization relationships between packages. Graphically a deployment diagram is a collection of vertices and arcs.

*Deployment diagram commonly contains*

* Nodes
* Dependency and association in one of 3 ways

1. To model Embedded System.

2. To model Client\server System

3. To model fully distributed system



Fig:College Bus Tracking Deployment

**7. Implementation**

**7.1 Introduction:**

Implementing an object-oriented design generally involves using a standard object oriented programming language (OOPL) or mapping object designs to databases. In most cases, it involves both.

**Implementation using Programming Languages:**

Usually, the task of transforming an object design into code is a straightforward process. Any object-oriented programming language like C++, Java, Smalltalk, C# and Python, includes provision for representing classes. In this chapter, we exemplify the concept using C++.

**7.2 Introduction to Activity Diagram:**

An Activity diagram shows the flow from activity to activity. An activity is an ongoing non-atomic execution within a state machine. An activity results in some action results in a change of state or return of a value. Action encompasses calling operation, sending a signal, creating or destroying objects, or a pure computation such as evaluating some expression.

Activity diagram commonly contains

1. Activity states and action states

2. Transitions

3. 0bjects, it may contains nodes and constraints.

Activity states and action states

An executable atomic computation is called action state, which cannot be decomposed. They rendered as lozenge shape. Activity state is non atomic, decomposable and take some duration to execute.

Transition:

It is the path from one state to the next state, represented as simple directed line.

Branching:

When alternate paths exist, branching arises which is represented by open diamond. It has one incoming transition, two or more outgoing transitions.

Forking and joining:

The Synchronization bar, when splits one flow into two or more flows is called Fork. When two or more flows are combined at synchronization bar, the bar is called Join.

Swim lanes:

Grouped work flow is called swing lane. All groups are portioned by vertical solid lines. Each swim lane specifies Locus of activities and has a unique name. Each swim lane is implemented by one or more classes. Transition may occur between objects across swim lanes.



Fig:Bus Activity diagram

**8. Testing**

**8.1 Introduction to Testing:**

Testing is the process to find any deviation from the expected working of the system. If there is no deviation from the expected behaviour of the system then the project is successful otherwise failure. Testing can't be done in a full-fledged manner because of the time and budget constraints.

**Black box and white box testing:**

Black box testing: This tests the inputs and the corresponding outputs.

White box testing: The code where exactlythe fault occurs can be identified by thismeans of testing.

**Unit Testing:**

Unit testingindividual unitsdetermine if they are fit for use. A unit is thesmallest testable part of an application. Inprocedural programming a unit may be anindividual function or procedure. Unit testsare created by programmers of occasionally by white box testers.

**Integration Testing:**

Integration testing is the activity of finding faultswhen testing the individual components together. Structuraltesting is the culmination of integration testing involving all the components of the system.

**System testing:**

System testing tests all the components together, seen as a single system to identify faults with respect to scenarios from the problem statement and the requirements and design goals identified in the analysis and system design.

**8.2 SAMPLE TEST CASES**

**LOGIN FORM:**

|  |  |  |  |
| --- | --- | --- | --- |
| SL.NO | TEST CASE | EXPECTED RESULT | TEST RESULT |
| 1. | Enter Valid name  and password &  click on login button | Software should display  Main window | Successful |
| 2. | Enter invalid details | Software should not display  main window | Successful |

**CHECK AVAILABLE BUSES TEST CASE:**

|  |  |  |  |
| --- | --- | --- | --- |
| SL.NO | TEST CASE | EXPECTED RESULT | TEST RESULT |
| 1. | Click on Available  buses button | List of Buses will be  displayed | Successful |
| 2. | Click on Available  buses button | No Buses will be  displayed | Successful |

**TRACK TEST CASE:**

|  |  |  |  |
| --- | --- | --- | --- |
| SL.NO | TEST CASE | EXPECTED RESULT | TEST RESULT |
| 1. | Click on track | Bus location will be  displayed | Successful |
| 2. | Click on track | Bus passed by | Successful |

**STOP UPDATES TEST CASE:**

|  |  |  |  |
| --- | --- | --- | --- |
| SL.NO | TEST CASE | EXPECTED RESULT | TEST RESULT |
| 1. | Click on stop upda-  tes | Updates will be  stopped | Successful |

**9. ER-DIAGRAM AND DATABASE DESIGN**

**E-R DIAGRAM:**

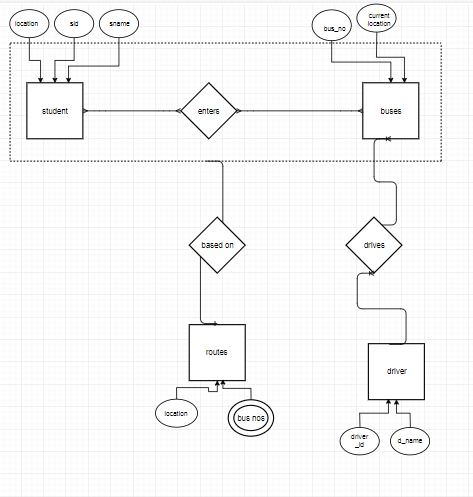


Fig: E-R Diagram of Online College Bus Tracking System

**DATABASE DESIGN:**

**TABLE NAME: STUDENT**

|  |  |  |
| --- | --- | --- |
| NAME | TYPE | ATTRIBUTE |
| sid | Int(15) | Primary key |
| sname | Varchar(25) | Not Null |
| location | Varchar(25) | Foreign key and Not Null |

**TABLE NAME:** **BUSES**

|  |  |  |
| --- | --- | --- |
| NAME | TYPE | ATTRIBUTE |
| busid | Int(15) | Primary key |
| Current location | Varchar(25) | Not Null |

**TABLE NAME: Driver**

|  |  |  |
| --- | --- | --- |
| NAME | TYPE | ATTRIBUTE |
| driver\_id | Int(15) | Primary key |
| d\_name | Varchar(25) | Not Null |

**TABLE NAME: Routes**

|  |  |  |
| --- | --- | --- |
| NAME | TYPE | ATTRIBUTE |
| Location | Varchar(25) | Primary key |
| Busno’s | Int(15) | Primarykey |

**10.CONCLUSION**

The project titled “ONLINE COLLEGE BUS TRACKING SYSTEM using GSM and satellite communication” is a model for vehicle tracking unit with the help of gps receivers and GSM modem. Vehicle(BUS) Tracking System resulted in improving overall productivity with better fleet management that in turn offers better return on your investments. Better scheduling or route planning can enable you handle larger jobs loads within a particular time. Vehicle tracking both in case of personal as well as business purpose improves safety and security, communication medium, performance monitoring and increases productivity. So in the coming year, it is going to play a major role in our day-to-day living. We have completed the project as per the requirements of our project. Finally the aim of the project i.e. to trace the vehicle is successfully achieved.

**11.REFERENCES**

http://www.pace.ac.in/documents/ece/COLLEGE%20BUS%20TRACKING%20SYSTEM%20USING%20GPS%20AND.pdf

http://1000projects.org/college-bus-tracking-system-java-project.html

http://javapapers.com/android/android-location-tracker-with-google-maps

http://www.codeproject.com/Articles/665527/A-GPS-Location-Plotting-Android-