Designing an Application for the Public Transport Management

Adrian LUPASC *

ABSTRACT
Smart mobile devices are being used increasingly, especially among young people who frequently use them, for ever more diverse purposes. The number of mobile apps has increased drastically in a relatively short period. Thus, mobile apps can turn a smart phone into an e-reader, payment instrument, planning resource, game console, and more. Therefore, we can say that smart mobile devices and, implicitly, their specific applications are increasingly part of our daily lives. For this reason, the purpose of this paper is to present how to design an application for smart mobile devices that provide public transport information in a city. This application can be of real use to both those who live in that city and those who do not live in that city. In this context, the paper presents three UML (Unified Modeling Language) diagrams describing both the processing of the future information system and the conceptual structure from the perspective of data that this application will handle.

© 2018 EAI. All rights reserved.

JEL Classification: C99, I29, J15, L63, L86
Keywords: UML diagrams, Class diagram, Use case diagram, Activity diagram, Public transport application

1. Introduction
Designing an information system is always an important activity because it is the basis of further developments that will be carried out in order to identify the specific processing of the future application. Clearly, the design stage has the analysis as its starting point, where a fundamental foundation is represented by the elaboration of the specifications to be solved. Choosing the optimal design method will be done in accordance with the objectives of the future IT system, the nature of the intended processing, as well as the programming language that will be the basis of the application development.

Thus, the fundamental objective considered in this paper is to design an application for smart mobile devices that provides information on public transport in a city. The design and implementation of such an information system generates a number of advantages and benefits, the most important being:

- fast access to information on public transport (stations and route map, route timetable) on any mobile device that has this application installed, without the need for an Internet connection;
- the possibility of searching for the means of transport running between two stations or between two streets that the user inserts into the search module;
- the opportunity to run the app on both Android and iOS smartphone mobile devices.

2. UML language
UML is a language for specifying, building, viewing and documenting components of an object-oriented computer application. Choosing this modeling language to underpin the design of a mobile application that manages public transport in a city is due to the fact that it generates a number of advantages: “independence from the design method, independence from programming language, communication improvement, facilitating reuse and managing complexity” [Georgescu, 2002, p. 165].

Underlining language-specific aspects does not exclude specifying how to use them. UML assumes that the methodology is “guided” by use cases, which it is based on the architecture of the system, and the process of applying the methodology is iterative and incremental.

UML is a visual modelling language and can be defined briefly as a language for viewing, specifying, building and documenting models. Its utility and applicability on a large scale results from the fact that it is an open standard, achieves the whole life cycle of software development and covers many types of applications. Visual modelling is based on five main characteristics: it discovers the processing that takes place within the system, using case-usage analysis; is a good means of communication; simplifies / reduces system complexity; defines the architecture of the software; allows reuse of components.

* Dunarea de Jos University of Galati, Romania. E-mail address: alupasc@ugal.ro
2.1. Use case Diagram

The Use Case diagram is a description in a structured language of a potential situation that an application may or may not solve. This diagram describes how an actor in the reality to be modeled interacts with the organization or the system as a whole. The Use Case diagram allows you to view the actors, their actions, and the boundaries of the future system. Each scenario must produce an observable result for one of its actors. Actors in low-level subsystems are internal to them and are not explicitly defined.

The specific elements of a usecase diagram are as follows:
- the actor - an actor is, in principle, a system user, but it can also be another computer system that interacts with the system being analysed;
- use cases - are represented in the form of an ellipse inside which the name of the scenario is written;
- associations - are used to indicate the relationship between an actor and a use case, in the sense that the actor participates in some way in that scenario.

Thus, Figure 2.1 shows the use case diagram of a computer system for managing public transport in a city.

Thus, in accordance with the Use case diagram of the proposed system presented in Figure 2.1, the actors and their interactions with the system are established, as well as the links between scenarios and actors. Scenarios are basically the dynamic part of the application and reflect the behaviour of the IT system that will be implemented.

The only actor interacting with the system is the user of the public transport application. He has, in the first instance, three possibilities: to press the Help button to find out how to use the application, to see details of a particular route, or to find out which route / routes run between the two points introduced. The user can view details of a particular route, and for this one must select a type of transport means, which is highlighted in the diagram by the include type dependency between Select routes and Select means of transport scenarios routes. Then he has to select a preferred route, which is modelled by the include type dependency between Select transport mode and Select route scenarios. In this way, the user will be directed to the user interface that will contain details of that route.

View route details is an expandable scenario (extend type of dependency) with the view route map scenario, an option that could be extremely useful for any user. Also, as outlined in the diagram in Figure 2.1, this scenario can also be extended with the view route timetable scenario, thus giving you the opportunity to view a route timetable when it is not known by the user of the application.

The user will also be able to search for a route that runs between two points of the city, this being represented in the chart by the association between the actor and the Select “Search” scenario. The search can be expanded by selecting the preferred type of search. This way, the user can enter the city’s street names or station names as the starting and ending points, which is modelled by an extend type of dependency between the Select “Search” and Modify Search types scenarios. The user must enter the departure and arrival points according to the selected search type, and the application will display, as appropriate, one, several or no routes, as highlighted in the Use Case diagram through the extend type dependency between the
Search Data Entry and Show route scenarios. If there is at least one route, the user will also be given the opportunity to view the details of the route(s).

2.2. Activity Diagram

The activity diagrams are part of the group of behavior diagrams and show the chaining of workflows from one activity to another. Each activity is a processing of a class in the system. The activity diagram shows the dynamics of the system, its purpose being to provide a picture of the coordination of internal flows, independent of the action of the external actor.

The activity diagram describes the activities and responsibilities of the elements that make up the system and can be used for the following purposes:

- modeling the internal behavior of a method (implementation of an operation), this being the most used case when using this type of diagram;
- analyzing the interactions of a Use case activities;
- illustrate the way in which several Use case are organized and the links between them.

A chart of activities is an extension of a status diagram, which means that such a chart will have all the features of a status diagram, but also new ones. The elements of a chart of activities are: action states, transitions, block of decisions, and timing blocks.

In this regard, Figure 2.2 shows the activity map associated with the mobile application for public transport in a city.

As can be seen in Figure 2.2, from the initial state of the system, the user has the choice of viewing the details of a particular route (stations, map, timetable) and searching for a route that runs between two points (stations or streets) of the city. Thus, in the activity diagram, the transition that starts from the initial state and to which the Select action is associated, enters into a decision block, making it possible to opt for one of the two options.

![Activity Diagram](image)

Figure 2.2. Activity diagram for the proposed system
If the Routes option is selected, the user will be directed to the user interface that includes the three modes of transport (tram, bus and trolley), where he will need to select the desired transport means. He will then reach the interface where the previously selected transport means are displayed, and where it will need to select the route whose details he wishes to view. After selecting a route, he will be directed to the route details, where he will view the stations of that route as well as their respective streets. Here, the user can either select the View Map option that will lead him to a new screen where he will view the route map or select the View Timetable option to reach another screen where he can view the schedule in detail for the previously selected route.

If the Search option has been selected, the user will reach the search screen. Here he will have to enter the place he wants to leave and where he wants to go, these places can be stations or streets in the city, depending on the type of search chosen. Then he has to click on the search button. If a route is found, it will be displayed; otherwise the user can do a new search with other data. For the route found, the user can view the details of the route, as well as the map and its timetable.

### 2.3. Class Diagram

The design stage also involves the realization of the class diagram that has the role of presenting the classes in the system and the relationships between them (inheritance, composition, association). The class diagram is considered the pillar of the entire object-oriented modeling and presents what the system should do, its functions and how it will be built. Thus, in Figure 2.3, there is a diagram showing classes for tracking a route.

![Figure 2.3. Class diagram for the proposed system](image)

The AbstractLevelMediator class is an extensive abstract class of the parent main class, as well as the main classes of all the other graphic user interfaces. Practically, this class will have the task of displaying graphics on the screen of your mobile device. The data members of the class are:

- _assetManager – is an instance of the manager that deals with uploading resources;
- _levelProxy – is an instance of the proxy of this module;
- _mainMC – represents the screen graphic;
- _levelViewComponent – is the container where the graphics are added.

The methods of this class are the following:

- levelInitialized() – calls the main class of the search module after all initialization have taken place;
- getID() – method that returns the id of the screen;
load() / unload() – sends the proxy to load/download the necessary resources;
displayLevel() – this method is called to go from one screen of the application to the other.
AbstractLevelProxy class is also an abstract class which can be extended by the proxies of each application screen. It has the role of uploading / downloading the necessary resources to an application screen. The class has the following data:
AbstractLevelProxy class is also an abstract class which can be extended by the proxies of each application screen. It has the role of uploading / downloading the necessary resources to an application screen. The class has the following data:
The class includes the following methods:
- getIsLoaded() – returns true value if the screen has been loaded and false otherwise;
- getIsInitialized() – returns true value if the screen has been initialized and false otherwise;
- load() / unload() – has the role of loading/unloading the resources necessary for the screen;
- onLevelLoadComplete() – is called when the resources are finished loading.
IRunnable interface is implemented by the main classes of the application screen and includes the following methods:
- init() – will perform all necessary initializations before displaying a screen;
- start() – this method will be called immediately after initializing and displaying the screen;
- sleep() – the method is called when exiting the application without closing it, to keep the application in its last state;
- wakeup() – method called when returning in the application and resuming it from the last point;
- destroy() – method called when exiting the screen to clean the memory of the resources that aren’t needed anymore.
The BackButton class is responsible with displaying the back button. The member data of this are:
- _aInteractive – is an instance of the class which deals with the application buttons;
- _graphics – represents the graphics of the button;
- _initialScale – is the initial scale of the button. It will be scaled according to the resolution of the screen on which the application will run.
The methods of this class are the following:
- onAInt() – the method is called when the button is pressed;
- setVisible() – the button visibility is set;
The SearchScreenProxy class extends the AbstractLevelProxy class and has the role of uploading data about all existing routes to the XML file. The only member data of this class is _routes, which is a RouteVO type vector that will store all track information. The methods of this proxy are as follows:
- init() – uploads data from the XML file
- getStations() – returns the stations of the route received through routeNumber parameter;
- getStreets() – returns the streets on which the stations of the route are, returned through routeNumber parameter;
- getType() – returns the means of transportation received through routeNumber parameter;
The RouteVO class has the role of storing the data about the route. It has the following data members:
- _id – variable that stores the number of the route;
- _type – represents the type of transport mean on a route;
- _stations – is an Array variable which stores the stations of the route;
- _streets – an Array type of variable that stores the streets of the route.
The SearchTypeView class has the role of managing the search type chosen by the user for two buttons associated to the graphic interface: Stations and Streets.
The RouteFoundView class has the role of displaying the route found after a search. Its main data members are:
- _transportTF – is the textfield in which the transportation means is displayed for the found route;
- _routeNumberTF – is the textfield in which the number of the found route is displayed;
- _detailsTF – is the textfield in which the details of the found route are displayed.
The main class of the search module is SearchScreen, and the most important member data are the following:
- _logicXML – represents the XML file which contains the logic of the search module
- _backBtn – is an instance of the back button;
- _routesFound – is a RouteFoundView type vector;
- _searchTypeView – is an instance of the SearchTypeView class;
- _searchBtn – represents the search button in this screen.

3. Conclusions and final remarks
The number of smart mobile devices users (phones and tablets) has increased a lot in recent years. Taking this into consideration, mobile applications for such devices have increased and application areas are becoming more diverse.
Mobile applications can be very useful to smartphone owners because they can access some information anytime, anywhere. For this reason, a mobile app that provides information on public transport in a city could be extremely useful for people who live in that city and for those who live outside the city.

In this context, in this paper I have presented the design method (using three UML diagrams) of an application that will aim to manage public transport in a city.

References