Technical University of Moldova
Inginerical department S.A.

Report

№: 6

AMOO

Subject: Statechart Diagrams. Domain Analysis. SOLID Principles

Author: Terman Emil FAF161
Prof: R. Melnic

Chisinau 2018

Tasks

- model the application using 3 **Statechart Diagrams**;
- perform the Domain analysis of your project;

Theory

Statechart diagrams

Statechart diagrams describe different states of a component in a system. The states are specific to a component of a system. Such a diagram describes a state machine. State machine can be defined as a machine which defines different states which are controlled by external or internal events.

Here are the main purposes of using *Statechart Diagrams*:
- to model the dynamic aspect of a system;
- to describe different states of an object during its life time;
- define a state machine to model the states of an object;
- to model the life time of a reactive system;

Domain Analysis

Domain analysis is the process of analyzing related software systems in a domain to find their common and variable parts. It is a model of wider business context for the system. Domain analysis produces domain models using methodologies such as domain specific languages, feature tables, facet tables, facet templates and generic architectures, which describe all of the systems in a domain.

SOLID Principles

The term SOLID is a mnemonic acronym for five design principles intended to make software designs more understandable, flexible and maintainable.
- **S**ingle responsibility principle: a class should have only a single responsibility (i.e. changes to only one part of the software's specification should be able to affect the specification of the class);
- **O**pen for extensions and closed for modification;
- **L**iskov substitution principle: "objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program";
- **I**nterface segregation principle: "many client-specific interfaces are better than one generalpurpose interface";
- **D**ependency inversion principle: one should "depend upon abstractions, **not** concretions";

GRASP Principles

The different patterns and principles used in GRASP are controller, creator, indirection, information expert, high cohesion, low coupling, polymorphism, protected variations and pure fabrication. All these patterns answer some software problems, and these problems are common to almost every software development project.

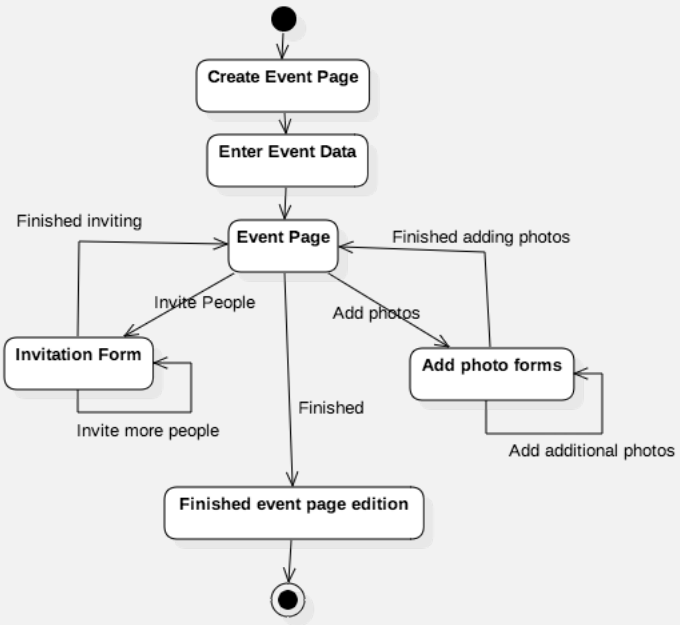
These techniques have not been invented to create new ways of working, but to better document and standardize old, tried-and-tested programming principles in object-oriented design.

Statechart diagrams

In *Figure 1* is represented the Statechart of creating a new event. First the user must enter the event details, then he is able invite people or add event photos. When he is done, the user is redirected to the event's View.

Figure 2 is about adding a new photo. The user is able to load a local image or indicate a link. If the file or the link is invalid, the user is returned to the initial state. When successfull, the loaded photo is shown in full screen. From here, it's possible to tag people, add a location or a comment. After submit is pressed, a corresponding message is displayed. In this state, it's possible to add more photos.

In *Figure 3* is shown the statechart of a friend request. First, the user tries to find some people in the search forms. If nothing satisfactory is found, he can cancel the action. Then profile is selected. It's also possible to cancel the action from this state. Here, a



Img 1: Create an event

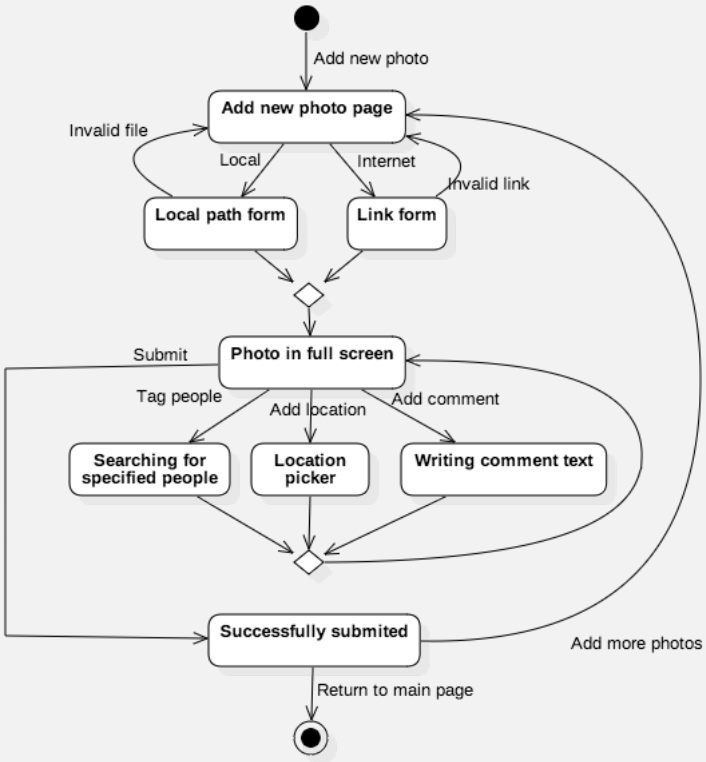
friend request button is available, if pressed, a message is displayed and the user is redirectioned to a page with people he may know. He can select other users or finish the action.

Domain analysis

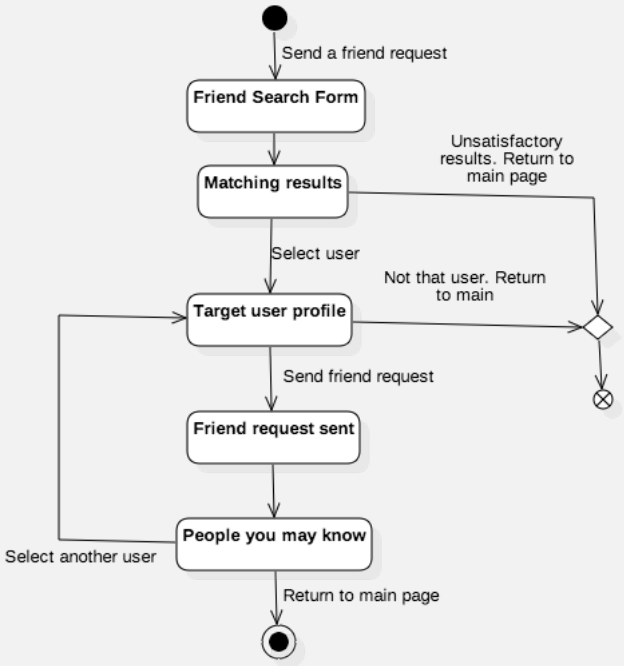
- this platform is from the social networking **domain**. People are continously looking for a faster, intuitive and safier app, so I think, my application's main advantage is the full comunication privacy. Another strong point is that this project will be available as open source;
- *The importance of this theme*: a user's privacy is often secretly ignored, or at least, the user isn't 100% sure his chats are private. So, it would be a good idea to make the project open source so that other programmers can confirm my project's privatness;
- **Twiter** and **Instagram** are **similar applications**, but the user's private content is used for advertistment targeting and other nasty things. Instagram is more media based platform, whereas Twiter, it's more focused on comunication;
- *The purpose and objectives*: This app is supposed to be open source, meaning that there will be many comunity contributions, which will greatly affect the popularity.

Conclusion

In this laboratory work I have learned how Statechart diagrams can be useful in real life production. It is another great way of explaining to non programmers what are the main ideas of a project. During this lab I took a quick glimpse to *Domain analysis*. While making this analysis, I came to the conclusion that my app needs more features to have a chance to survive on the market. So I came up with the idea of making it an open source project. Which acutaly may work out...



Img 2: Add new photo



Img 3: Friend request