Bank

Analysis and Design Document

Student: Sorin Urcan

**Group: 30432**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 23/apr/15 | 1.0 | <details> | Urcan Sorin |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

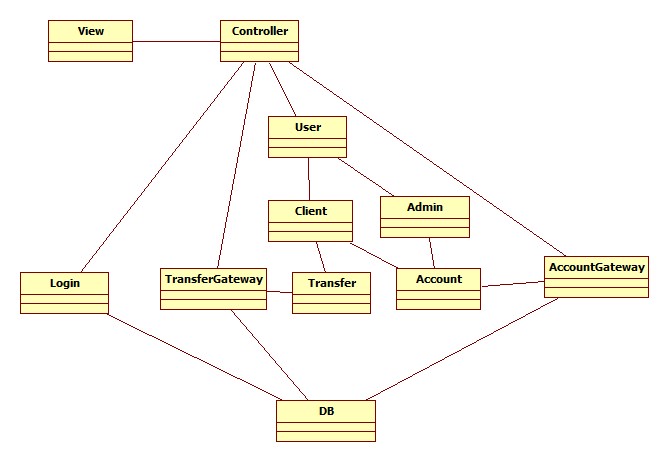
VI. Bibliography 5

# Project Specification

# Use Swing/C# API to design and implement an application for the clients and employees of a bank. The application should have two types of users: a regular user which would be a client of the bank and an administrator user represented by an employee of the bank. Both users have to provide a username and a password in order to use the application. The regular user can perform the following operations: - View all accounts (account information: identification number, type, amount of money, date of creation). - Transfer money between accounts. - Pay utilities bills. - Generate reports containing account transactions in a particular period. The administrator user can perform the following operations: - CRUD on client’s information (name, identity card number, personal numerical code, address, etc.). - Create/update/delete/view clients' accounts. - Generate reports for a particular period containing the activities performed by a client.

# Elaboration – Iteration 1.1

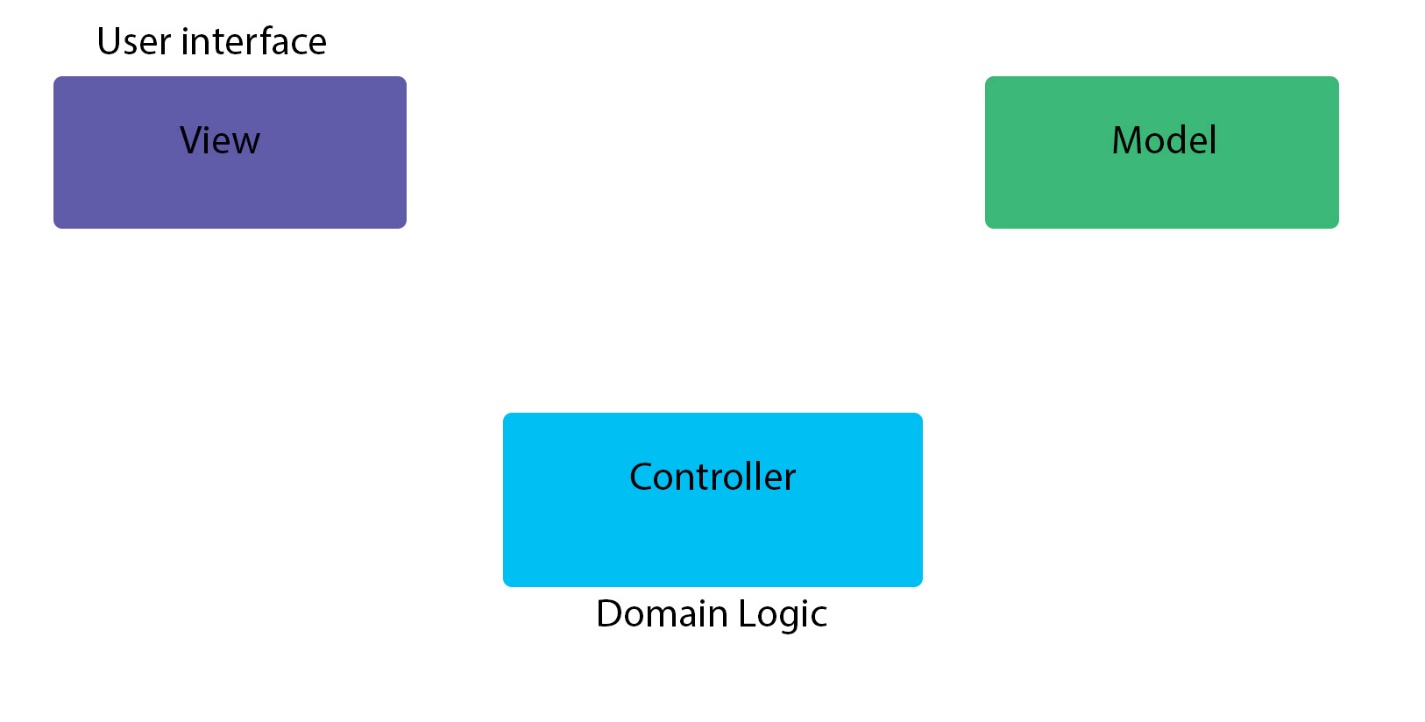
# Domain Model



# Architectural Design

## Conceptual Architecture

The chosen architectural pattern is the model-view-controller. It fits the requirements very well, since it divides the software into three main parts, the model, the view and the controller. For our system, the model is represented by the User, Client, Admin, Transfer, Account, the view is represented by the user interface and the controller is the domain logic of the system.

These three components interact with each other. The controller can send commands to the model to change its state, or send commands to the view in order to update its presentation of the model. The view requests information from the model through the controller. By interacting with the view, the user sends commands through the controller, which can modify the model or the view. 

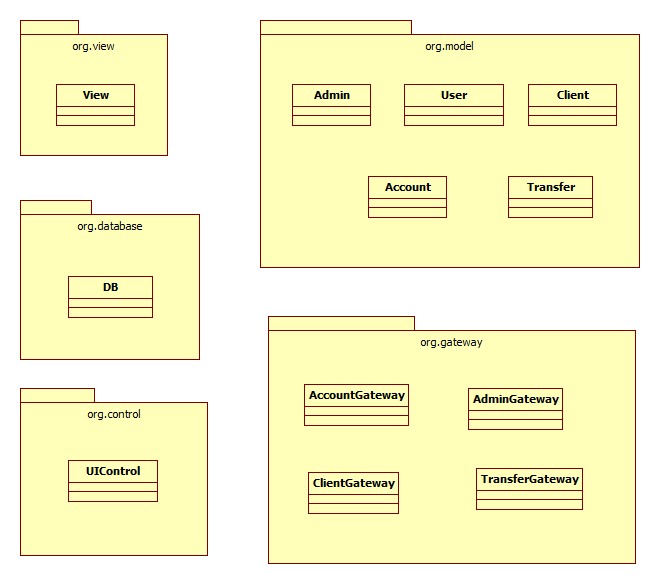
The architectural style is client-server based. This facilitates storage of large amounts of data, data which can be accessed remotely. The database would contain several tables, for holding user account information, client information, transactions.

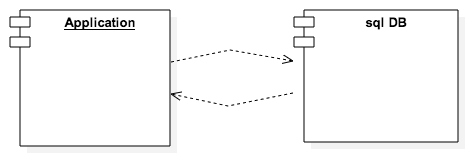
Patterns used in the domain logic are: Gateway.

The Singleton pattern restricts the instantiation of a class to a single object. In our implementation it is useful in instantiating only once the Database connection.

## Package Design

## Component and Deployment Diagrams



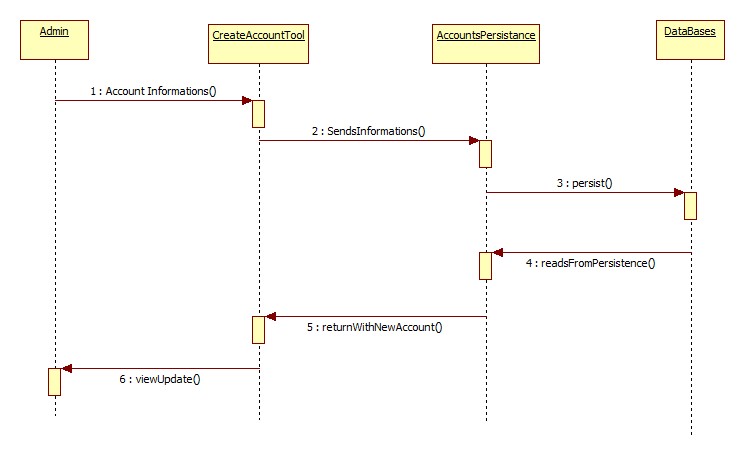


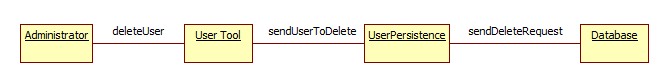
# Elaboration – Iteration 1.2

# Design Model

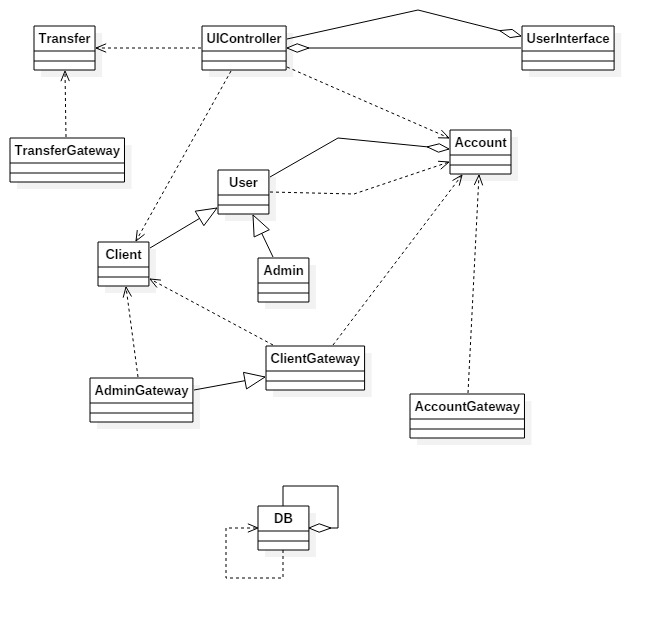
## Dynamic Behavior

System dynamics is an approach to understanding the behavior of [complex systems](http://en.wikipedia.org/wiki/Complex_system) over time. It deals with internal feedback loops and time delays that affect the behavior of the entire system. What makes using system dynamics different from other approaches to studying complex systems is the use of feedback loops and stocks and flows. These elements help describe how even seemingly simple systems display baffling [nonlinearity](http://en.wikipedia.org/wiki/Nonlinearity).





## Class Design



# Data Model

Data modeling is the formalization and documentation of existing processes and events that occur during application software design and development. Data modeling techniques and tools capture and translate complex system designs into easily understood representations of the data flows and processes, creating a blueprint for construction and/or re-engineering.

The system data model is represented by classes that represent the abstraction of the business logic. The model layer contains the data models that are used in the system. The data model is in the lowest level of the system hierarchy. The client is modeled by the Client class that contains all the information this system needs in order to model the interactions of a client. The client account and application user data are modeled in their respective classes. All the classes that model data are simple containers for the data they hold and they do not process in any way the data they receive. These objects are passed between modules which modify and ultimately translate them in an SQL query where database interaction is needed.

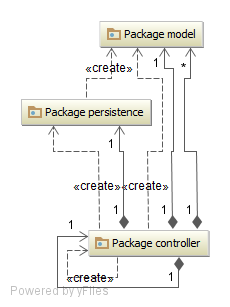
# Unit Testing

**Unit testing** is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for usage. Unit testing was not completed for this project scarce of time and resource limitations

# Elaboration – Iteration 2

# Architectural Design Refinement

The system will use the layered architectural pattern. The layered architectural pattern is used so that the system will be easier to extend and allows creating a decoupled system with parts that can be changed in case an upgrade is wanted (e.g. choose another persistence type). The main layers that will be present in the application will be: the Presentation layer which will be used by users to interact with the system. The Application Layer (controller) is the top level layer which is integrating all the layers below it. The layer on the lowest level will be the Business (model) layer, which integrates all business objects abstracted in classes.



# Design Model Refinement

The design model builds on the analysis model by describing, in greater detail, the structure of the system and how the system will be implemented. Classes that were identified in the analysis model are refined to include the implementation constructs. The design model is based on the analysis and architectural requirements of the system. It represents the application components and determines their appropriate placement and use within the overall architecture. In the design model, packages contain the design elements of the system, such as design classes, interfaces, and design subsystems, that evolve from the analysis classes. Each package can contain any number of sub packages that further partition the contained design elements. These architectural layers form the basis for a second-level organization of the elements that describe the specifications and implementation details of the system.

The original view on the system had in mind high cohesion, low coupling, polymorphism and good abstraction of data.

# Construction and Transition

Construction of the system was made incrementally with each of the layer being modeled from bottom up. After the lower layers were implemented the development concentrated on the next layer and so on.

Sometimes it was needed to refactor code from the lower level, but as the project progressed it became clear that a good design eliminates such problems.

# System Testing

Integration testing in such a system would be made using a mocking technique for the server part. We would send information through one component and expect the mock to receive some object. Then test that the object received matches the object we would expect. Testing in the reverse part is also possible, simulate an object being sent form the server to the client in the same manner.

# Future improvements

In a future release of this system administrators could have access over the entire database system. More validations could be performed for the fields that are in the forms in order to eliminate human error in the system.

# Bibliography

www.google.ro