News Agency App

Analysis and Design Document

Student: Razvan Pasca

**Group: 3043**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

This application is intended to serve a basic news agency company. It aims to provide a simple interface for its customers to access the published articles from a list and to enable its employees (writers) to quickly modify the contents of an article, publish a new one or delete an old one. It comes under the form of desktop apps, based on the client server architecture.

# Functional Requirements

The system presents different use cases, each having different functional requirements. The most important ones are presented below:

* The app must support 2 types of users: regular users (do not require login, can only read articles) and writers (who need to login to perform their specific activities)
* The app must concurrently support more users logged in, each having its own view of the articles.
* The articles must be refreshed automatically in case a modification is done by an author. This refresh must take place across all users displays, without needing their intervention
* A writer can modify any article, even if it hasn’t been written by himself

# Non-functional Requirements

For the proposed system, there are certain non-functional requirements which are important to be accomplished, given the use case scenario:

* Real-time feed of articles of the news agency, without significant delays (1 sec or less loading time)
* Supporting a large number of customers using the app in the same time
* Data integrity of the articles and the way information is stored in the system
* Scalability: in case the number of users is going to be prohibitively large, the system should be upgradeable to accommodate them. This would be mostly solved by increasing the hardware resources of the system (either horizontally or vertically).

2. Use-Case Model

Use case: Writer adds new article

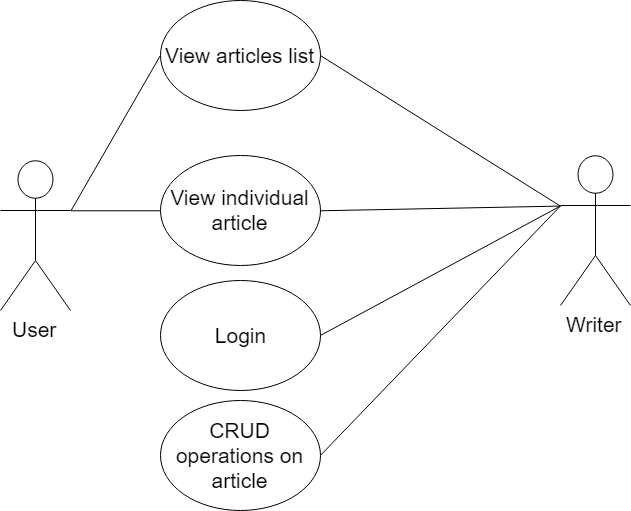
Level: User-goal level

Primary actor: Writer

Main success scenario:

Pre-conditions: writer is logged in

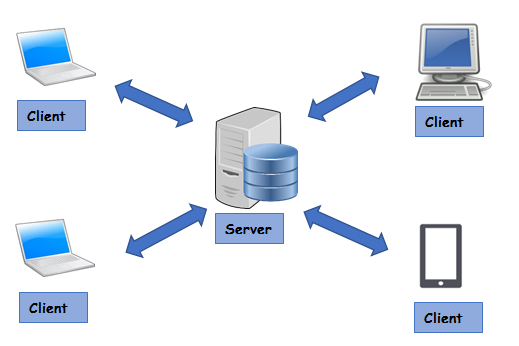
1. The writer clicks on an empty table row
2. A new window shows up with blank article fields which need to be completed
3. The writer enters the fields and optionally the author
4. The writer hits the add article button
5. The article is added to the database and automatically sent to the feed of the current users



3. System Architectural Design

**3.1 Architectural Pattern Description**

For this application, given its usage scenario, the client server architecture is the first choice. The basic functioning principles assume two distinct participants in the process: the server and the client. The client usually sends requests about desired information such as articles and the server answers accordingly. The server also hosts the database with the articles.



In our case, the communication is done through sockets. They allow communication between different processes which can be on different machines or on the same machine. The communication protocol is based on Unix file descriptors, i.e. the participants communicate by writing and reading from a hypothetical file. More specifically, stream sockets are employed, which guarantee the order of the items sent and return an error in case the communication is impossible.

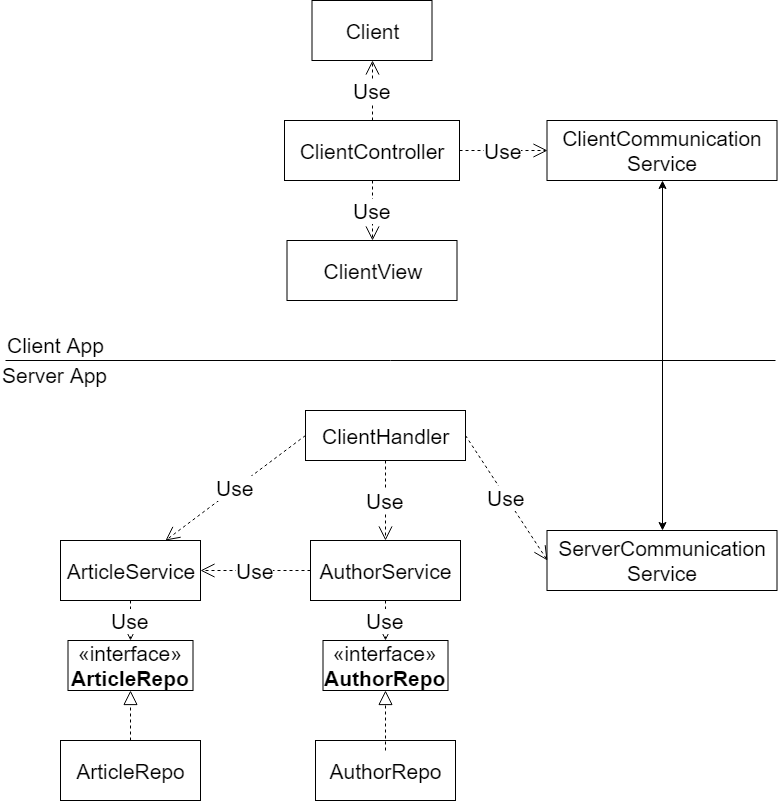
For improving the separation between the components of the system, layered architecture has also been employed, with the following layers: persistence, business and communication service.

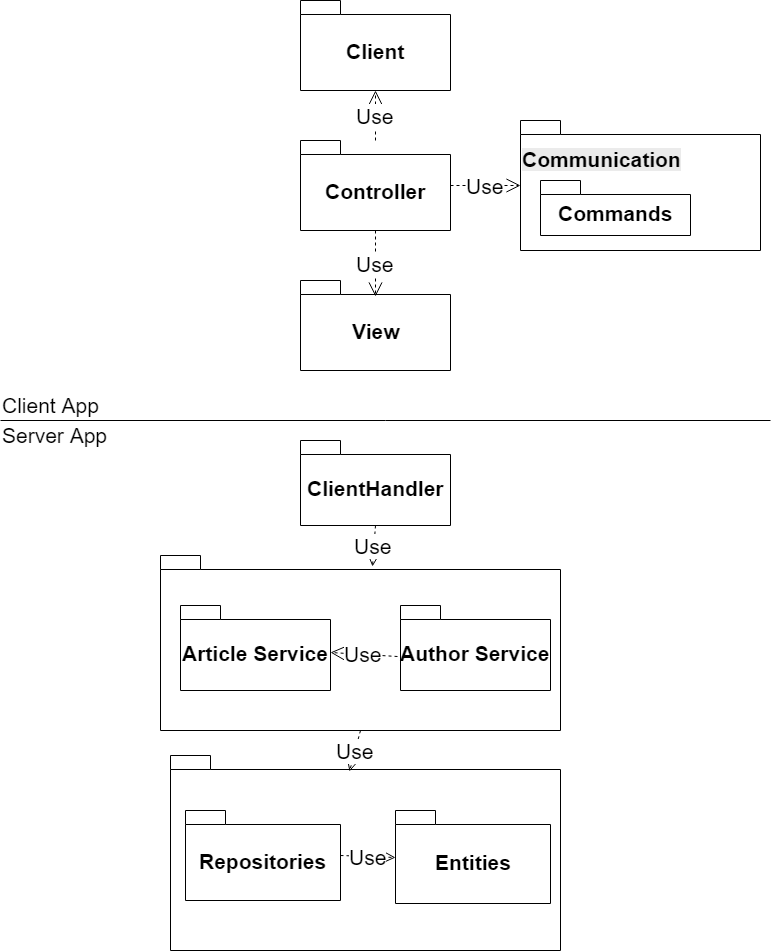
On the client side, we have a simple GUI together with a client controller and a communication service.

**3.2 Diagrams**

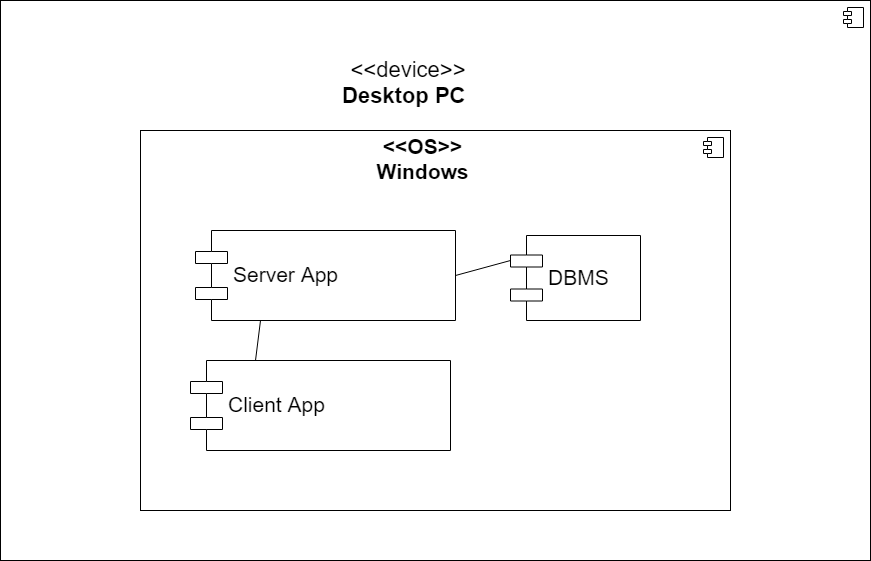
In the following pages the diagrams with the system architecture is presented, hiding the minor implementation details in order to get a better view of the overall aspect of the architecture and design logic behind it. We present the architecture diagram, the package diagram and the deployment diagram.

As underlined before, the layered architecture and the communication channel can be easily observed in the following diagram, which emphasizes the design philosophy.



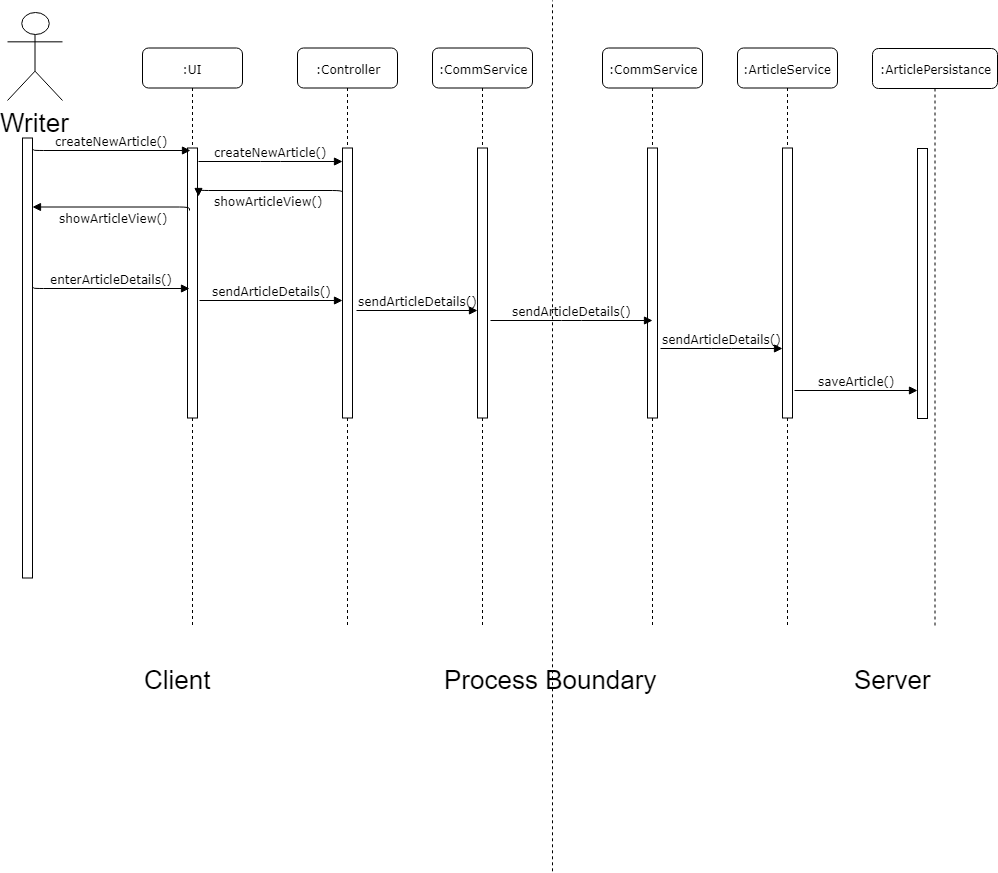


In the current case, both applications have been deployed on the same desktop computer. However, they can be easily deployed across several stations, keeping the server together with the DBMS and having an arbitrary number of clients connected.



4. UML Sequence Diagrams

In the following, the UML sequence diagram for the write article use case is presented, following the steps described in the previous chapters. The boundary between the 2 processes have been depicted, the communication being made by the communication services of the 2 apps. Each is responsible for interpreting and sending its own commands.



5. Class Design

In the actual implementation of the above presented templates, several design patterns have been employed, given the room of maneuver and possible approaches possible for the app. Also, given that the designed app is approximatively an event driven one based on the client server communication, patterns such as observer and command are a given.

**5.1 Design Patterns Description**

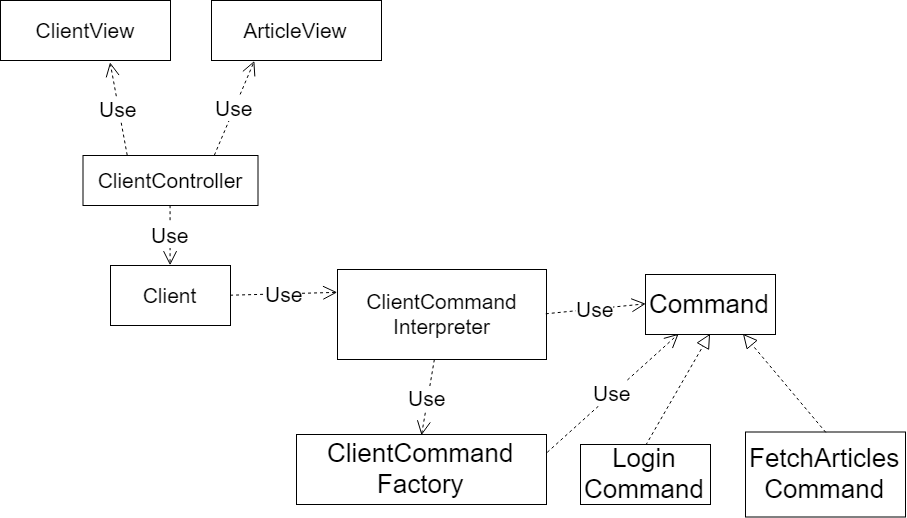
**Observer Design Pattern:** Applying the observer design pattern in this project is one of the objectives of the assignment. In the current case, the pattern has been applied twice. Once for broadcasting messages across all clients (the service was the observable and the observer were the client handler) when a CRUD operation has been performed. The other case of use is for updating the fields of the GUI, where the articles list inside the Client app is the observable and the GUI the Controller the observer. Once a modification on the article list is made, such as getting a new one from the server, the changes are immediately reflected in the GUI.

**Command Design Pattern:** The command design pattern has been applied for making the communication between the processes easier. In our case, the base class command is extended by concrete commands used by our application: LoginCommand, FetchArticlesCommand etc. The command encapsulates the action in a string message and optionally the objects passed between the processes.

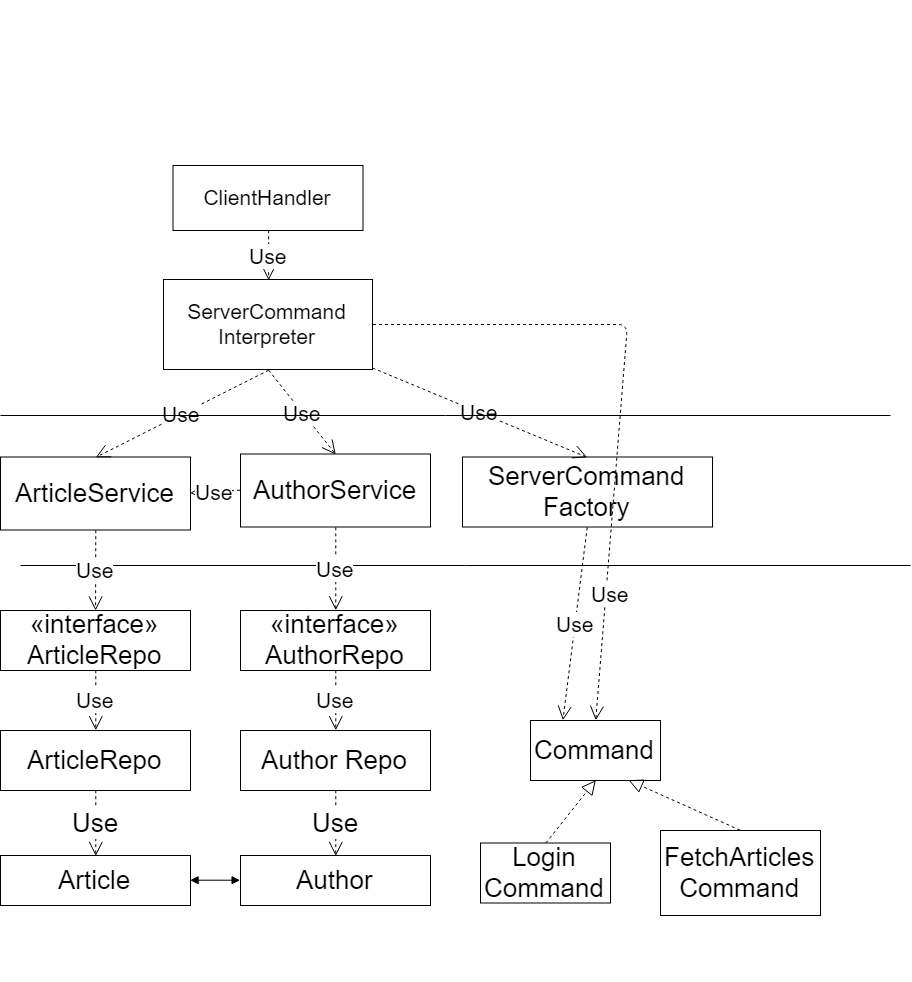
**Factory Design Pattern:** The factory design pattern has been applied in order to instantiate the commands based on their message string. Therefore, once a json command serialization arrives at one side of the socket, the factory takes that message and creates the concrete object corresponding to the intent and returns it to the class responsible for executing the commands.

**5.2 UML Class Diagram**

Client UML Class Diagram



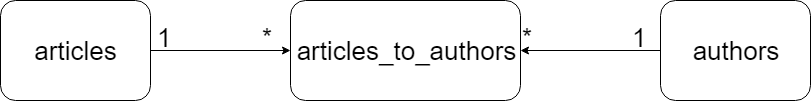
As it can be observed in the diagram, the main points of the client app are the controller and the command interpreter. The controller is a classical MVP one, having a reference both to the client model and to the views it controls. It is responsible for delegating different tasks across the entities of the system, through the client reference. The client delegates the command execution the command interpreter which receives the commands from the command factory. Both classes commit to the Command interface which is implemented by the different concrete commands used by our application. The Client class also implements the runnable interface, for enabling multithreading reading and GUI control.



As it can be observed from the server class diagram, we have employed the layer architecture when designing the app, distinguishing a persistence layer and a service layer which are used in waterfall by the client thread and the command interpreter. There exists a bidirectional association between articles and authors, each represented by a set collection, in order to map the many to many relationship. The command factory and the command interpreter have the same functioning principles as the ones in the client side of the app. Along designing the app, sticking to an interface and not an implementation as been one of the main guiding principles, taking care however not to overuse it. The low level entities are easily mapped on the database as it can be seen in the next chapter.

6. Data Model

Given the straightforward business logic of the app and the lack of actors and entities involved, the data model is also simple. We have a 1:1 mapping of the entities in the class diagram, with an added table for the many to many relationships. In our case, the Hibernate ORM has taken care of all the SQL queries, fetching and connecting to the database. This further improved the flexibility of the initial design while cutting development time significantly



Therefore, an article can have multiple authors which are automatically added by the app once an author edits/creates an article and an author can be credited for multiple articles. In case of entity removal, we delete the entry in the corresponding table and the entries in the linking table. This is also handled by the ORM mechanism inside Spring.

8. Bibliography

1. <https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm>
2. <http://javapeople.blogspot.ro/2006/07/what-are-some-advantages-and.html>
3. https://dzone.com/articles/design-patterns-command