MIDDLE EAST

TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

NewStreamLine

DETILED DESIGN REPORT

Ainura MADYLOVA	**e1408657**
Asiye KIYAK	**e1347673**
Fatma ÖZYILDIRIM	**e1347830**
Esin YÖNDEM	**e1348168**
Hüsna IŞIK	**e1347566**

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

The purpose of this document is to give the detailed design specifications of the NEWSUNI project of NewStreamLine. In this report, to have a major guideline for our future studies, we explained our design process in an illustrative way through our diagrams. These diagrams are Data Flow Diagrams (DFD) and Unified Modeling Language (UML) Diagrams. Through these diagrams, we tried to clarify the functional, structural and behavioral aspects of our system. Purpose of this project is to write the universal server for news exchanging that will give the ability to reach the news and posts via different types of protocols such as HTTP, SMTP, and NNTP using the secure data transfer layers. We have improved our initial design report, redefined some important models of our system, added new models to it and created some graphical user interfaces of the web page.

1.2 Scope and Project Description

NewStreamLine is a project that implements different access methods for news reading namely RSS, NNTP service, E-Mail list and Webpage. These different access methods are synchronized with the help of core. Core is the main part of our system including web service modules and extensions. From the user side when a user reply a post through forum page, post will feed RSS clients on the web, mailed to the subscribed users, displayed on a newsreader as a follow up and also displayed on the forum page.

1.3 System Overview

In this project, we aim to merge several service providers in NewStreamLine middleware. Among those providers, there are NNTP, SMTP, HTTP Servers and an XML file generator for RSS access. In order to implement a well-performing system, we decided to divide all that servers in particular models and proceed over them one by one.

First module is an NNTP server module that listens to the NNTP clients. As a server for NNTP data exchange, we have chosen Apache James 2.3.0 Server that includes not only an NNTP server but also an SMTP server as well. The reason for choosing James is that it is written in Java language, in other words James is platform independent. This module

communicates with client via appropriate news reader programs, and uses file repository as a storage type.

Second module is a Web server that provides an access to NewStreamLine via Internet. As a web server we used Apache Tomcat 5.0 and implemented web pages as Java Server Pages. Web Site has a forum-like structure that provides user with abilities of reading and writing post, subscribing to receive new posts via e-mail.

Third module is an SMTP server that automatically sends mails to users that subscribed for receiving posts of specified news groups via e-mail. This module communicates both with database and file system, taking user and mailing list from database and post list from file system.

Fourth module is SQL/XML service that provides connection with database and generates sql queries for insert, delete and update operations of database according to user's requests. These requests can come either from Web server module or from SMTP module.

Fifth module is File Query System module that builds a bridge between file repository and all other modules. It serves as a post list provider both for Web page and for SMTP module. It reports any changes occuring in file system to SMTP module that will process them and send to subscribed users.

To sum up, the Core connects each module to one another and provides data flow and repository synchronization. Since we record all incoming posts in one file system, there will be no need for reading posts and recording them to database. All services get the post data from file system.This makes our core work effciently.

To sum up, the Core waits for modules to communicate with each other and listens to the external ports, sends the related information in case of there is a need. The basic aim of this project is to make all of these services work together in synchronization and our Core application is created with this purpose of service.

1.4 Requirements

1.4.1 Hardware Requirements

As our project is mainly based on data storage all hardware requirements are focused on this topic. We have two kinds of data repositories to store all information of the system: one for NNTP server and one for SQL/XML service. Totally, we need 40 GB of Hard Disk Driver space and the same amount for the back up. Moreover, there must be an Internet connection fast enough to accept the daily load depending on the system usage.

1.4.2 Software Requirements

There are basically three servers for our system; NNTP, SMTP, HTTP Servers. We have used Apache Tomcat 5.0 web server for our project. We have used Apache James Server 2.0 for NNTP and SMTP. The Apache Java Enterprise Mail Server (a.k.a. Apache James) is a 100% pure Java SMTP and POP3 Mail server and NNTP News server. James is designed to be a complete and portable enterprise mail engine solution based on currently available open protocols. We have used MySQL Server 5.0 as a relational database management system for web page applications. For developing Java Server Pages, we are using IntelliJ Idea 5.1. on Windows machines and NetBeans 5.0. on Linux machines. Both of them use JDK 1.5.0. To connect to system via NNTP server, one should use any news reader (like Mozilla Thunderbird, etc), and to get RSS news feed, one should use any rss reader (like Aggregator, etc).

1.4.3 Functional Requirements

Visitor Functionality: A visitor can use NewStreamLine system only after subscription. Therefore, visitor can only subscribe to system in order to be a user.

User Functionality:

- Login to the system throughout the system dialog box,
- Create new topics/news groups,
- Subscribe/ unsubscribe to news groups,
- Post / reply to / read / discuss messages,
- Select news groups to be informed about via e-mail posting,
- Modify account information,

- See statistics about topic reading,
- Select different categories to be informed about,
- Search about a topic which already exists in the system.

Admin Functionality:

- Insert/delete news group,
- Insert/delete user,
- Delete posts,

1.4.4 Nonfunctional Requirements

Usability:

Companies from varying sectors and people from all age groups use Internet really extensively. Thus, our product gives service to a wide variety of users, such that people having not much knowledge about computers in profession can use our system. Thus, developing a user friendly interface and usability is very important for our project.

Security:

Security is the main concern of our product. There is a user authentication password-login system which provides a more secure working space for the system users. Moreover, the authentication is for maintaining the user hierarchy. Administrators have the right to modify the inputs and the outputs of the system, and manage the system database, modify it. Admin also has the right to change file system. Users do not have the right neither to reach to the system database, nor to modify it. One more important security aspect of our development is Secure Sockets Layer protocol, namely SSL. It is a high-level security protocol that protects the confidentiality and security of data while it is being transmitted through the Internet based on RSA Data Security's public-key cryptography.

Maintenance:

Maintenance is required to be able to overcome the problems occurred after enabling the system. With this aim, we have decided to obey to some standardization rules, such as the standardized variable names and producing codes throughout commenting. This is the

professional coding and if we do such coding, it will be available to enhance the system anytime if required. Therefore, to be a high-qualify project, the source code must be maintainable.

Reliability:

Our product will be so stable such that any minor problem will not cease the NewStreamLine NEWSUNI system. Moreover, we plan to do many tests including unit testing, integration testing and higher order testing after implementation.

2. DESIGN CONSTRAINTS

2.1 Time Constraints

Within next week we will be showing a prototype of the NewStreamLine that contains only few functionalities of main system. More important aspects as integration of SMTP server with NNTP server will be implemented in the second term. Web service and RSS feeding parts are nearly over. In February 2007 we probably will be dealing with the web page of our team. In March 2007 we will be integrating NNTP server with SMTP server. In April 2007 all security issues (as SSL) will be added to the system and we will start testing. By the end of May 2007 we will release the end product. In addition to all of these development stages, the aim of minimization of time consumption will remain our main concern.

2.2 User Interface

Graphical user interface of the system is represented by web page mainly. Despite the fact that, we are focused on implementing the core middleware, GUI is an important part of the project. Since all of our work, is shown through this interface, it must correctly demonstrate the abilities of the system. Even if we are not planning to spend much time on it, we will try to make it simple, user friendly, and functional. There can be found some user interface representations of web page through out this report.

2.3 Hardware Constraints

Our department will provide us with Linux Server Machine next semester. Since we do not have a server machine having the exact needed specifications, sometimes we have difficulties in implementing our project. Our machines have different operating systems, and we have no network among them. Consequently, it is difficult to work parallel with each other. Although

this semester does not require an intensive programming for the project, these constraints slowed us down to get more involved with the project.

2.4 Performance Constraints

NEWSUNI Project requires high network communication speed. Thus, the performance of the software is very important for us. Our system will use the existing servers and will not try to implement a new one, since it is time consuming and hard to finish in limited amount of time we have. This will avoid us from generating abuse network traffic and consume our time and bandwidth gap uneconomically.

3. DATABASE DESIGN

3.1 Database Tables

According to changes in our system, we have updated our database too. Since we save the incoming news in files, we have constructed file system instead of Post Table in database.
So, the constructed database in MySQL includes following tables.

User Table:

Name:	**Content:**	**Supplementary Information:**
ID	Int	Primary Key + AUTO INCREMENT
Password	Varchar(15)	Not Null
FirstName	Char(15)	
LastName	Char(10)	
Username	Char(20)	Not Null
E-mail	Char(40)	Not Null

Subscribe Table:

Name:	**Content:**	**Supplementary Information:**
User_ID	Varchar(10)	Foreign Key(ID)
Group_ID	Varchar(10)	Foreign Key

SendRead Table:

Name:	**Content:**	**Supplementary Information:**
User_ID	Varchar(10)	Foreign Key(ID)
Post_ID	Varchar(10)	Foreign Key

Mail_List Table:

Name:	**Content:**	**Supplementary Information:**
User_ID	Varchar(10)	Foreign Key(ID)
Group_ID	Varchar(10)	Foreign Key

3.2. SQL Commands of Database

In first step, we have specified the tables. The second step is constructing our database in MySQL by the following commands

```
create User(
ID int AUTO INCREMENT,
Password varchar(15) not null,
UserName char(20) not null,
firstName char(15),
lastName char(15),
E_mail char(40) not null,
primary key(ID)
);

create Subscribe(
User_ID varchar(10),
Group_ID varchar(10),
primary key(User_ID, Group_ID),
foreign key(User_ID  references User on delete cascade,),
);
```

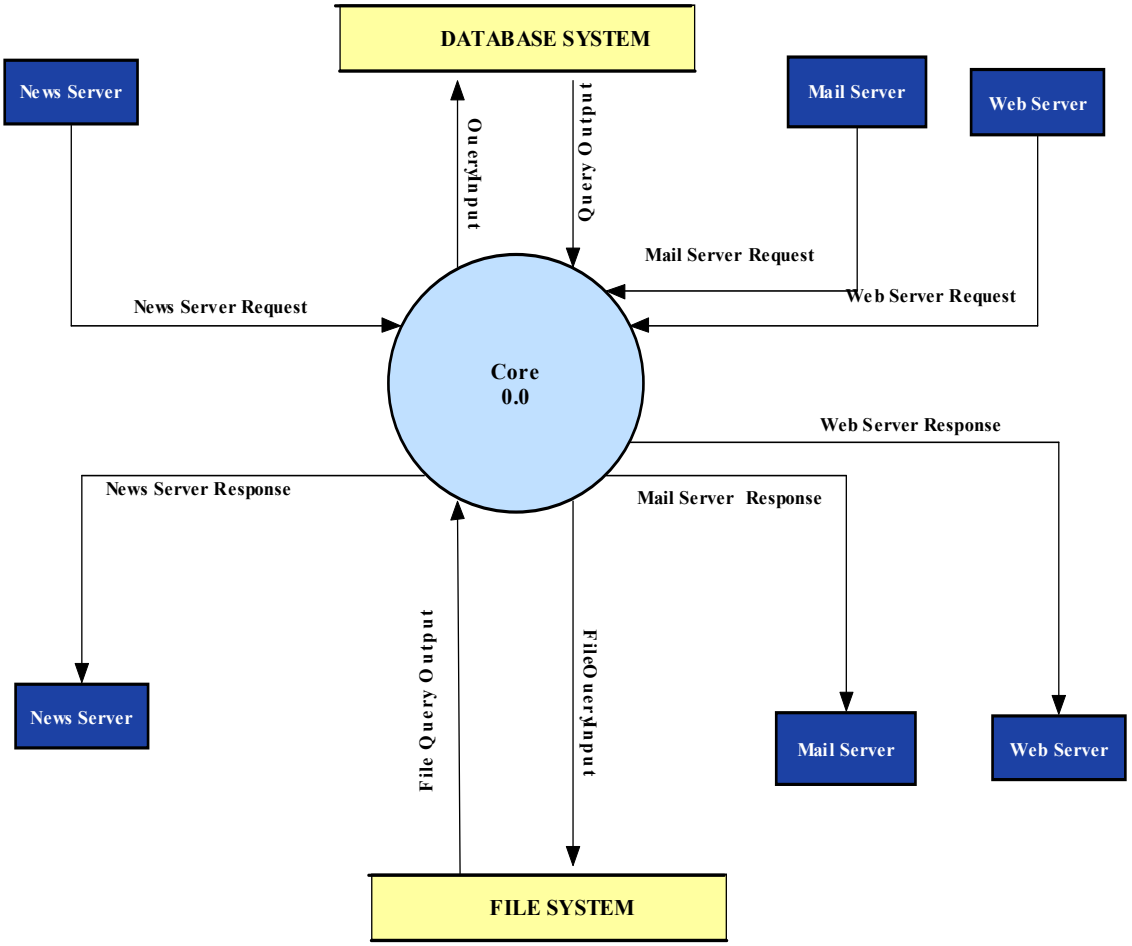
```
create SendRead(
User_ID varchar(10),
Post_ID varchar(10),
primary key(User_ID, Group_ID),
foreign key(User_ID  references User on delete cascade,),
);

create Mail_List(
User_ID varchar(10),
Group_ID varchar(10),
primary key(User_ID, Group_ID),
foreign key(User_ID  references User on delete cascade),
);
```

4. MODELLING

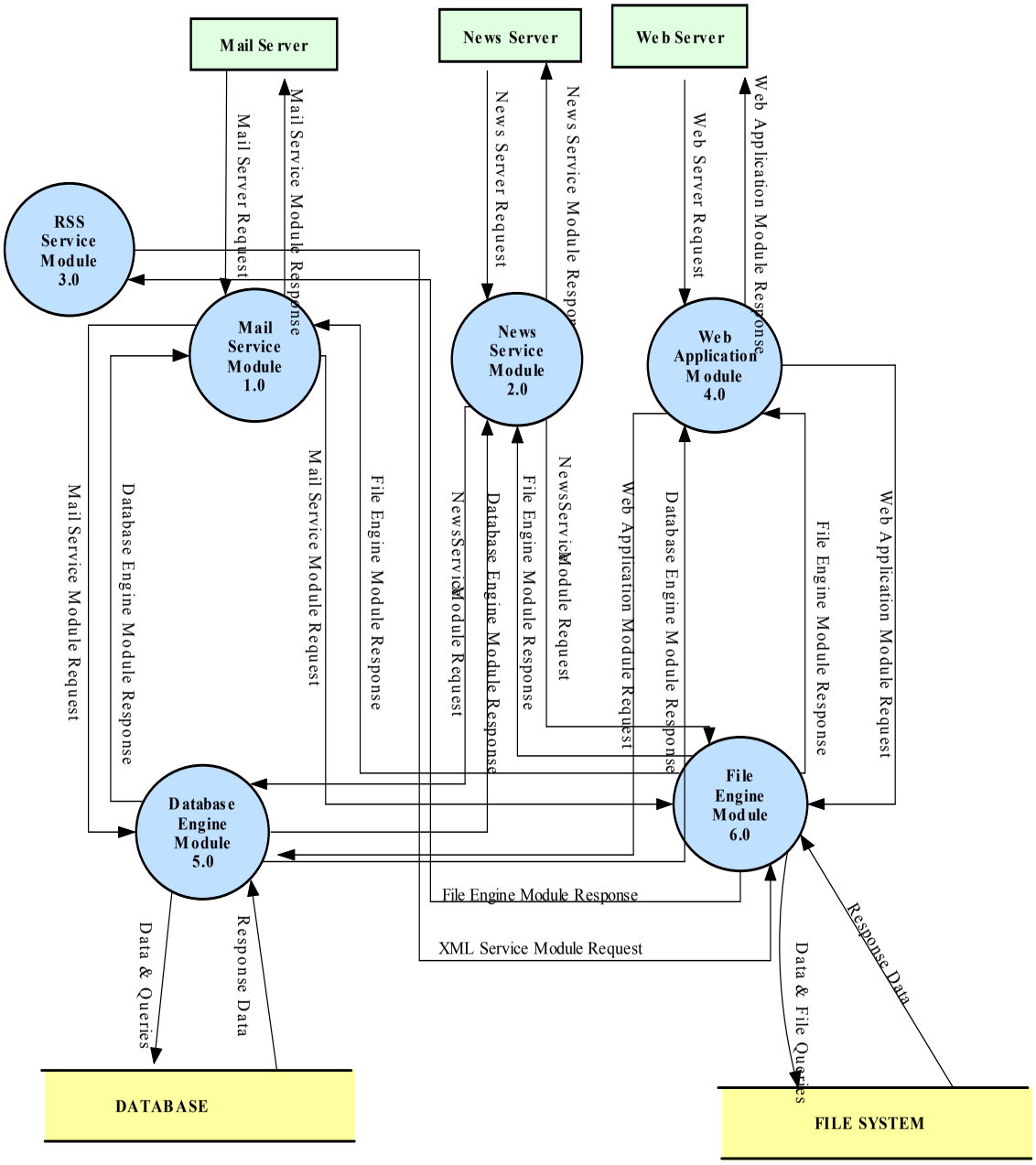
4.1. Functional Model

4.1.1 Data Flow Diagram - Core

Level 0

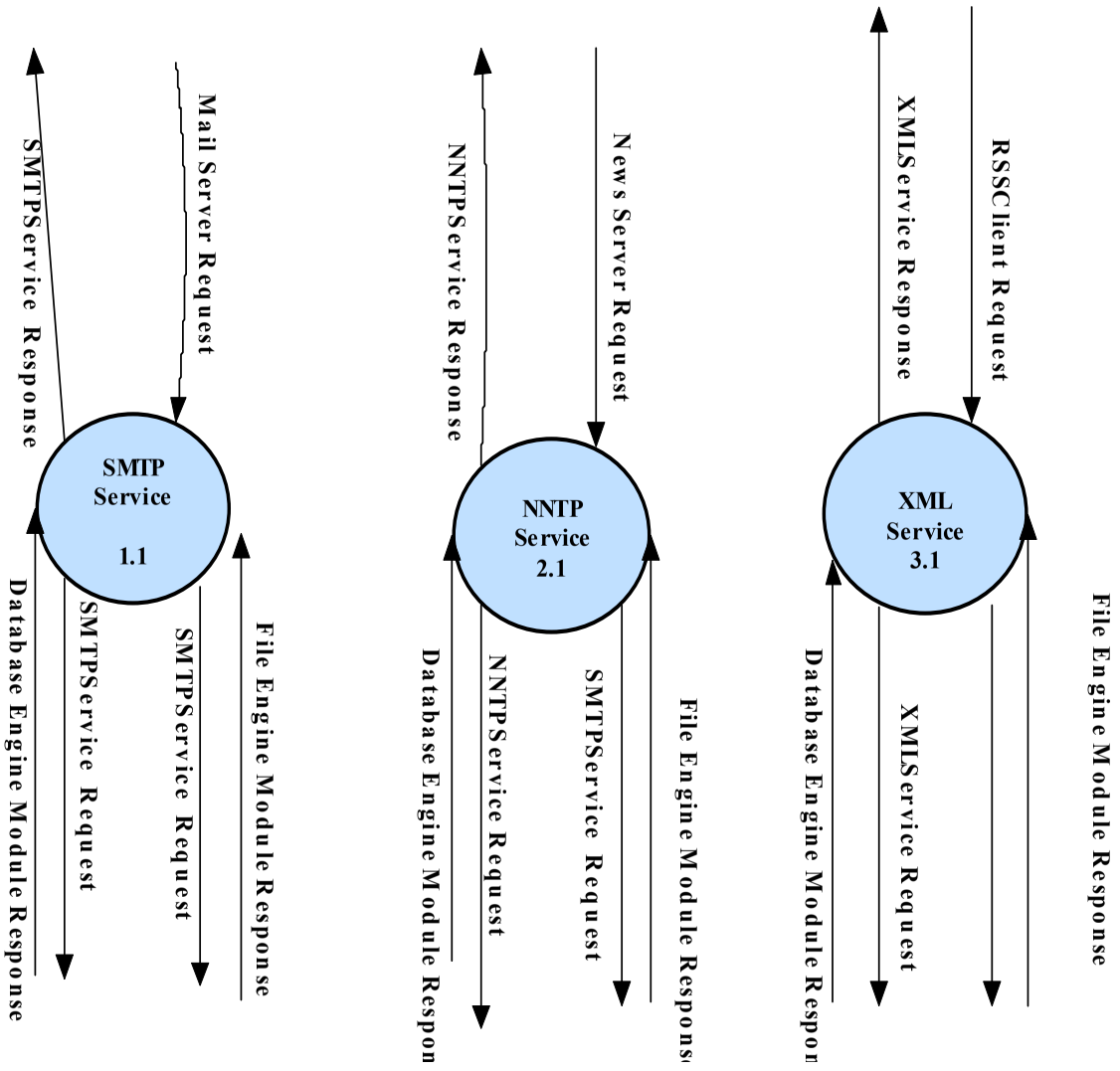


Level 1


Mail Server
News Server
Web Server
RSS Service Module 3.0
Mail Service Module 1.0
News Service Module 2.0
Web Application Module 4.0
Database Engine Module 5.0
File Engine Module 6.0
Mail Server Request
Mail Service Module Response
News Server Request
News Service Module Response
Web Server Request
Web Application Module Response
Mail Service Module Request
Database Engine Module Response
Mail Service Module Request
File Engine Module Response
NewsService Module Request
Database Engine Module Response
File Engine Module Response
NewsService Module Request
Web Application Module Request
Database Engine Module Response
File Engine Module Response
Web Application Module Request
File Engine Module Response
XML Service Module Request
Data & Queries
Response Data
Data & File Queries
Response Data
DATABASE
FILE SYSTEM

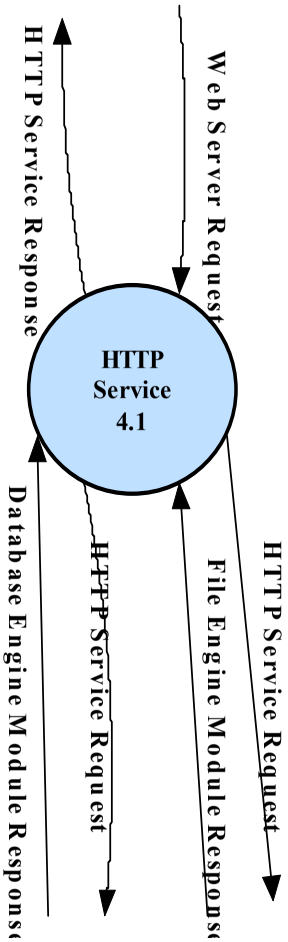
Level 2

Service Module



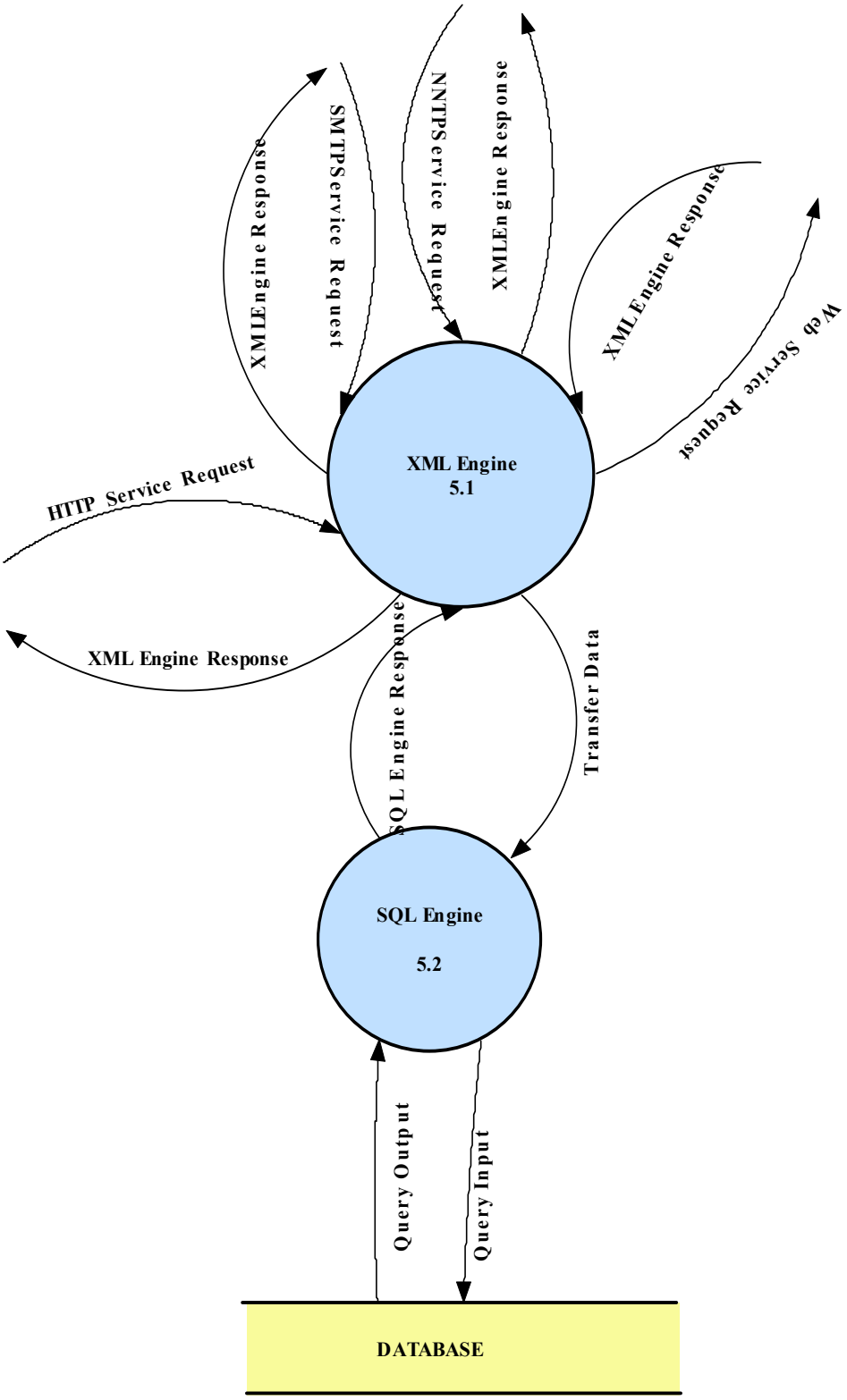
Level 2

Web Application Module



Level 2

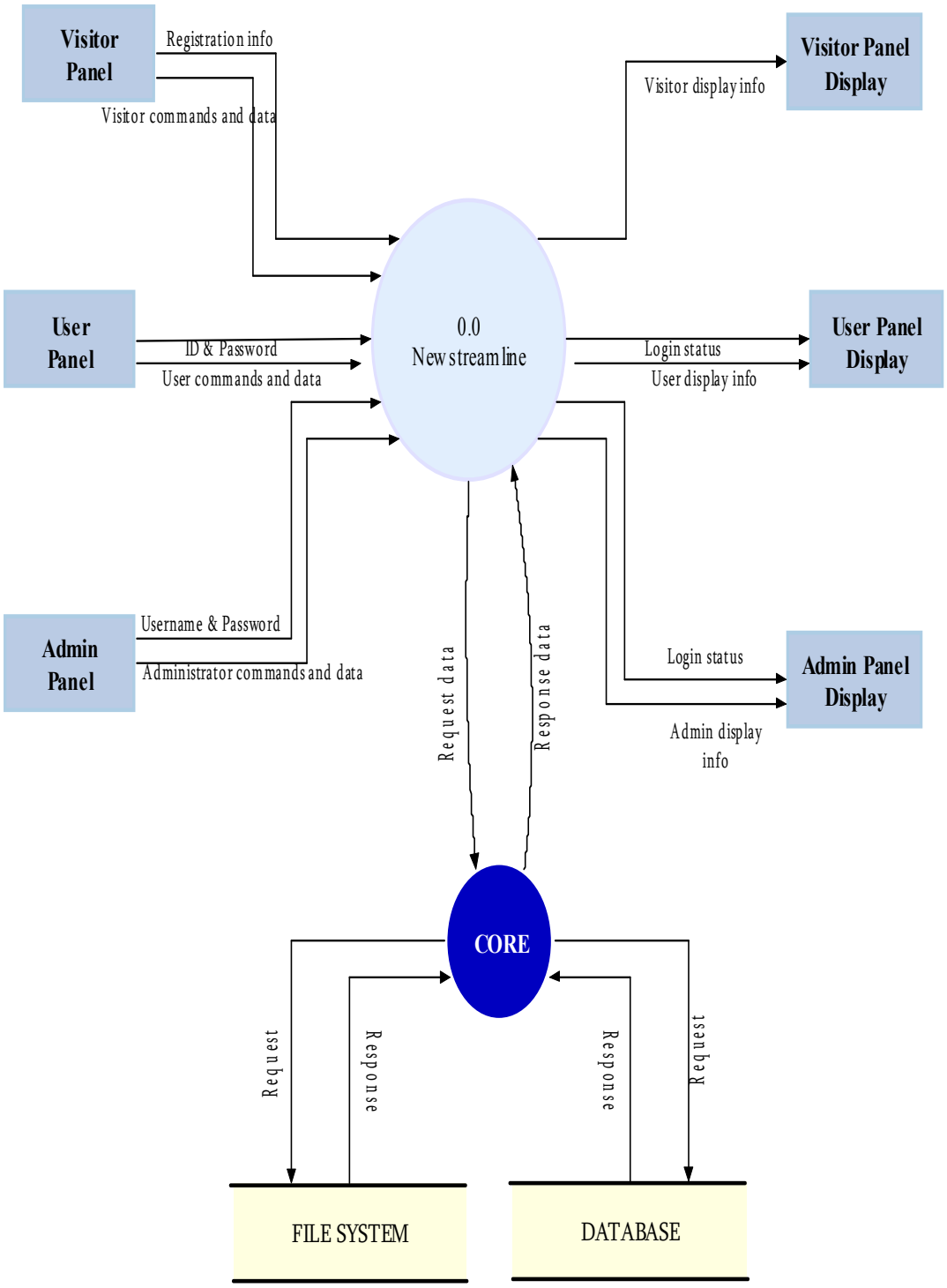
Database **Engine** **Module**



4.1.2 Data Flow Diagram – Web Application

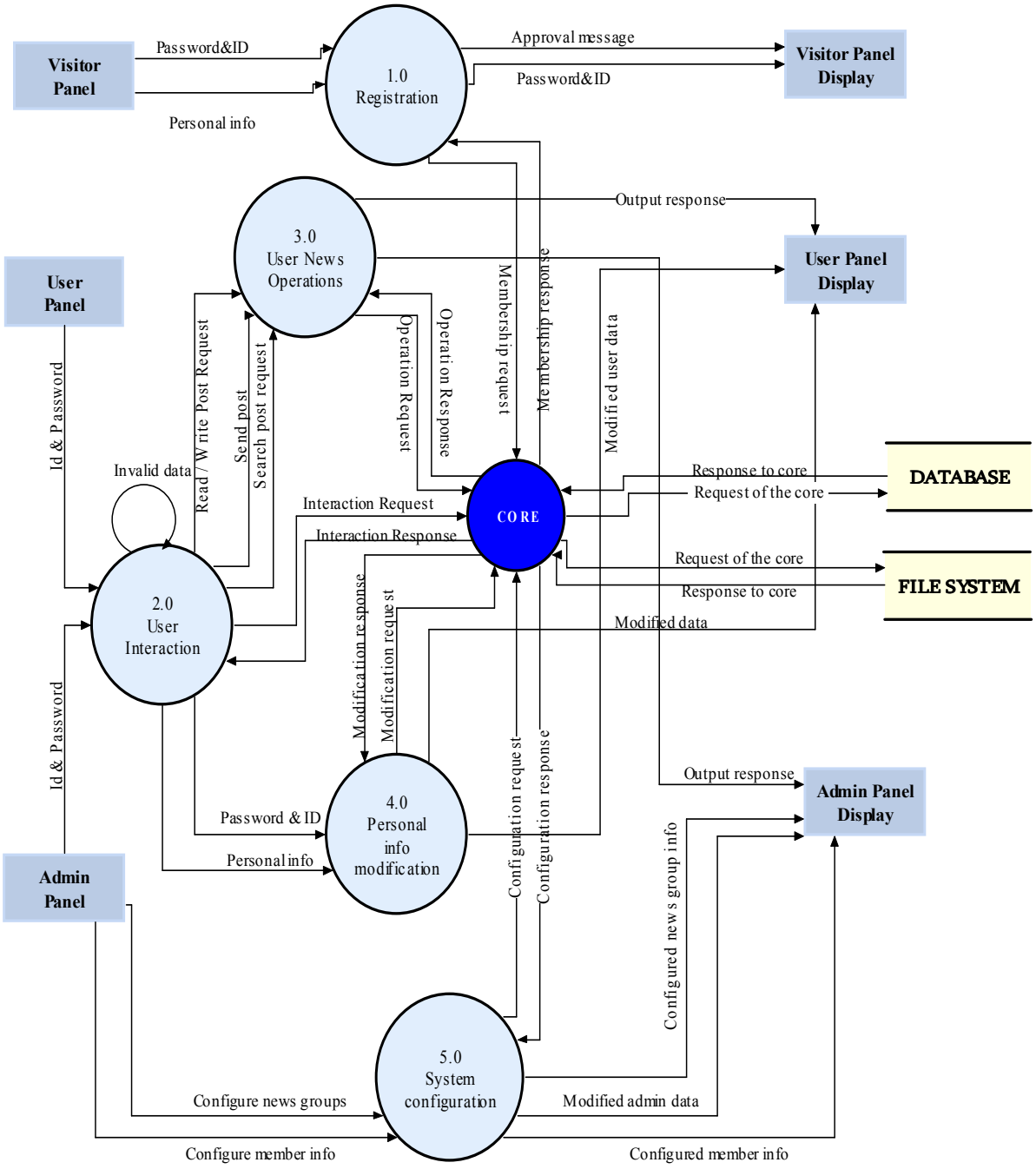
Level 0

Web Application



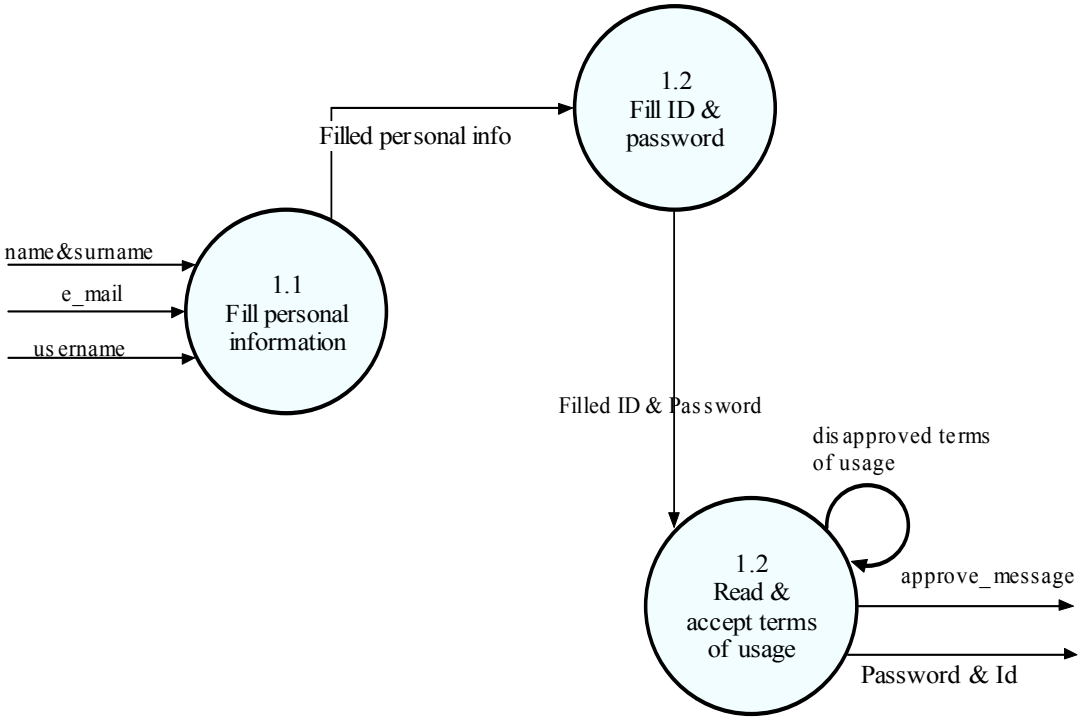
Level 1

Web Application



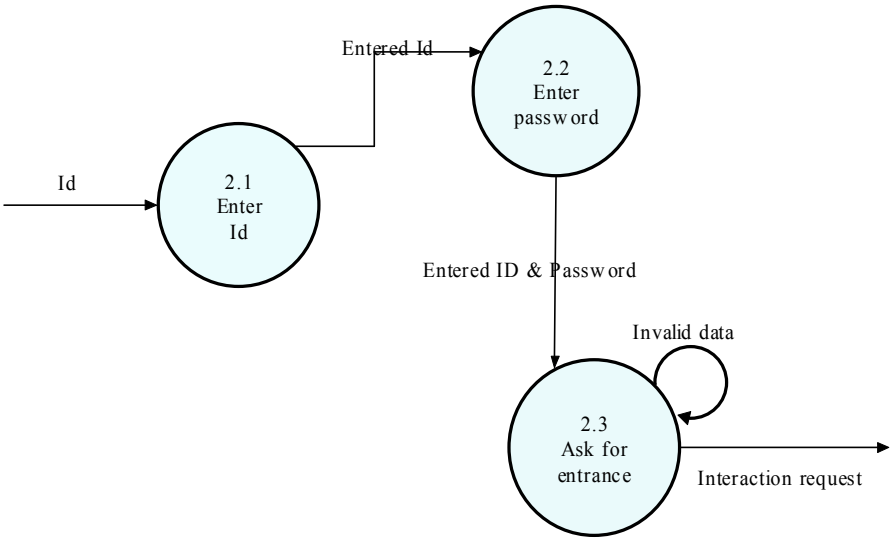
Level 2

Registration



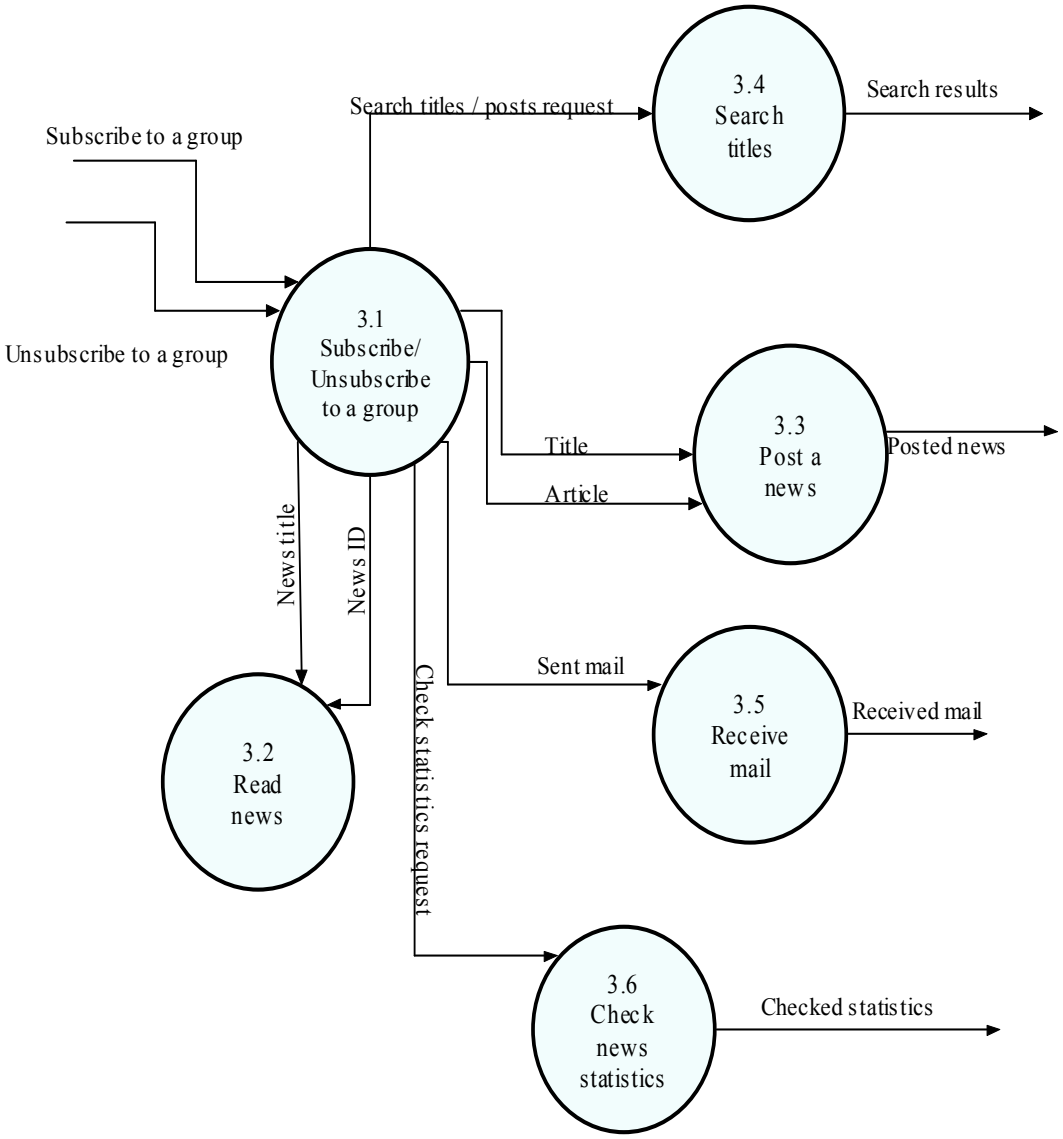
Level 2

User Interaction



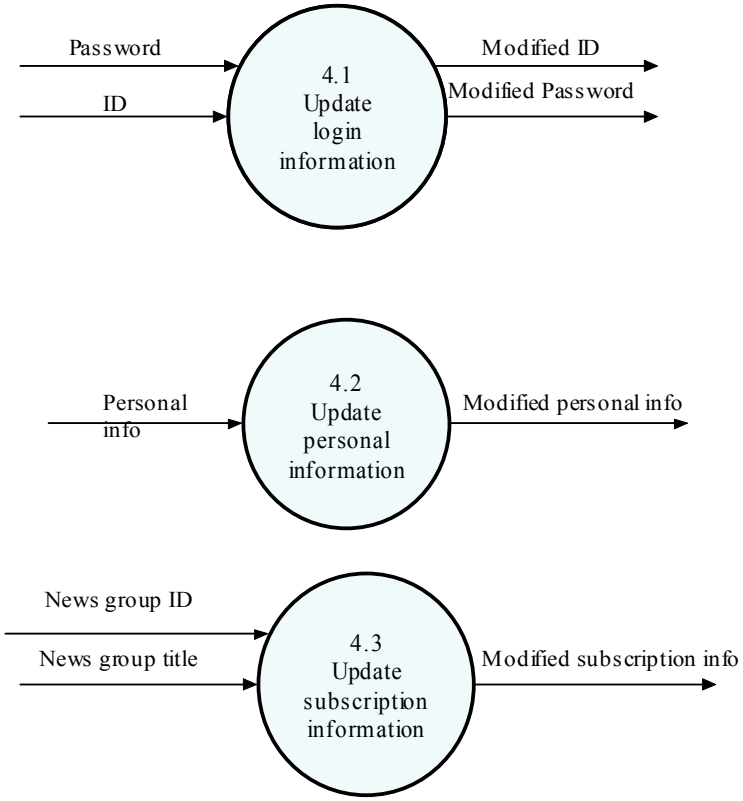
Level 2

User News Operation



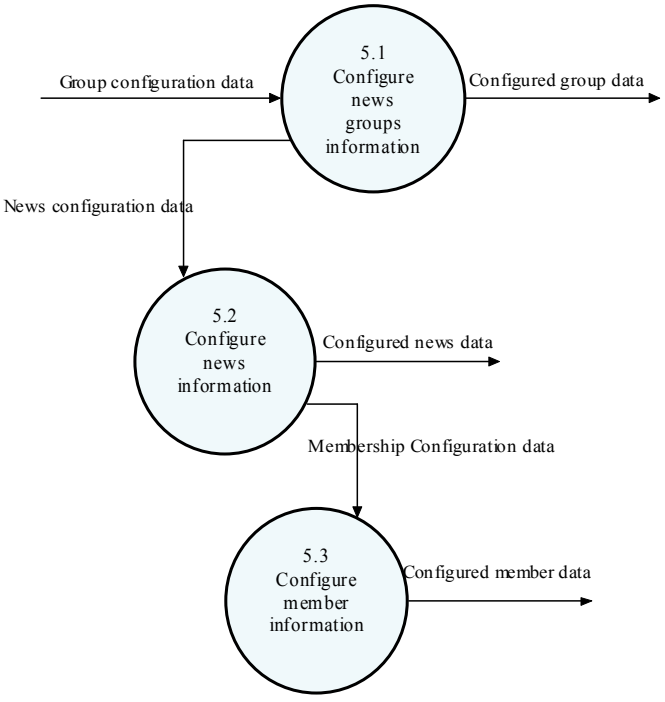
Level 2

Personal Info Modification



Level 2

News Server Configuration



4.2. Data Dictionaries

4.2.1 Data Dictionary of Core System

Name:	News Server Request
Alias:	None
Where & How It is used:	News Server Input
Description:	“sent requests by news server ”

Name:	News Server Response
Alias:	None
Where & How It is used:	News ServerOutput
Description:	“responses to news server ”

Name:	Mail Server Request
Alias:	None
Where & How It is used:	Mail Server Input
Description:	“sent requests by mail server ”

Name:	Mail Server Response
Alias:	None
Where & How It is used:	Mail Server Output
Description:	“responses to mail server ”

Name:	Web Server Request
Alias:	None
Where & How It is used:	Web Server Input
Description:	“sent requests by web server ”

Name:	Web Server Response
Alias:	None
Where & How It is used:	Web Server Output
Description:	“responses to web server ”

Name:	Query Input
Alias:	Query request
Where & How It is used:	Accessing database
Description:	“queries from core”

Name:	Query output
Alias:	Query response
Where & HowIt is used:	Database output
Description:	“Output data coming from database”

Name:	File Query Input
Alias:	File Query request
Where & How It is used:	Accessing file system
Description:	“ file queries from core”

Name:	File Query output
Alias:	File Query response
Where & How It is used:	File System output
Description:	“Output data coming from file system”

Name:	Mail Server Request
Alias:	None
Where & How It is used:	Mail Service Module Input
Description:	“Requests coming from mail server to mail service module”

Name:	Mail Service Module Response
Alias:	None
Where & How It is used:	Mail Service Module Output
Description:	"Responses coming from mail service module to mails server"

Name:	Mail Service Module Request
Alias:	None
Where & How It is used:	Database Engine Module Input
Description:	"Requests coming from database engine module to mail service module"

Name:	Database Engine Module Response
Alias:	None
Where & How It is used:	Database Engine Module Output
Description:	"Responses coming from mail service module to database engine module"

Name:	Mail Service Module Request
Alias:	None
Where & How It is used:	File Engine Module Input
Description:	"Requests coming from file engine module to mail service module"

Name:	File Engine Module Response
Alias:	None
Where & How It is used:	File Engine Module Output
Description:	"Responses coming from mail service module to file engine module"

Name:	News Server Request
Alias:	None
Where & How It is used:	News Service Module Input
Description:	"Requests coming from news server to news service module"

Name:	News Service Module Response
Alias:	None
Where & How It is used:	News Service Module Output
Description:	"Responses coming from news service module to newss server"

Name:	News Service Module Request
Alias:	None
Where & How It is used:	Database Engine Module Input
Description:	"Requests coming from database engine module to news service module"

Name:	Database Engine Module Response
Alias:	None
Where & How It is used:	Database Engine Module Output
Description:	"Responses coming from news service module to database engine module"

Name:	News Service Module Request
Alias:	None
Where & How It is used:	File Engine Module Input
Description:	"Requests coming from file engine module to news service module"

Name:	File Engine Module Response
Alias:	None
Where & How It is used:	File Engine Module Output
Description:	"Responses coming from news service module to file engine module"

Name:	Web Server Request
Alias:	None
Where & How It is used:	Web Application Module Input
Description:	"Requests coming from Web server to Web Application module"

Name:	Web Application Service Module Response
Alias:	None
Where & How It is used:	Web Application Module Output
Description:	"Responses coming from Web Application service module to Web server"

Name:	Web Application Module Request
Alias:	None
Where & How It is used:	Database Engine Module Input
Description:	"Requests coming from database engine module to Web Application module"
Name:	Database Engine Module Response
Alias:	None
Where & How It is used:	Database Engine Module Output
Description:	"Responses coming from Web Application module to database engine module"

Name:	RSS Service Module Request
Alias:	None
Where & How It is used:	File Engine Module Input
Description:	"Requests coming from file engine module to rss service module"

Name:	File Engine Module Response
Alias:	None
Where & How It is used:	File Engine Module Output
Description:	"Responses coming from rss service module to file engine module"

Name:	Data & Query Input
Alias:	Query request
Where & How It is used:	Accessing database
Description:	"queries from database engine module"

Name:	Response Data
Alias:	Query response
Where & How It is used:	Database output
Description:	"Output data coming from database"

Name:	Data &File Query Input
Alias:	File Query request
Where & How It is used:	Accessing file system
Description:	" file queries from file engine module"

Name:	Response Data
Alias:	File Query response
Where & How It is used:	File System output
Description:	“Output data coming from file system”

Name:	Mail Server Request
Alias:	None
Where & How It is used:	SMTP Service Input
Description:	“Requests coming to smtp service”

Name:	SMTP Service Response
Alias:	None
Where & How It is used:	SMTP Service Output
Description:	“Response coming from smtp service”

Name:	SMTP Service Request
Alias:	None
Where & How It is used:	SMTP Service Output
Description:	“Requests coming from smtp service

Name:	Database Engine Module Response
Alias:	None
Where & How It is used:	SMTP Service Input
Description:	“Responses coming to smtp service

Name:	SMTP Service Request
Alias:	None
Where & How It is used:	SMTP Service Output
Description:	“Requests coming from smtp service

Name:	File Engine Module Response
Alias:	None
Where & How It is used:	SMTP Service Input
Description:	“Responses coming to smtp service

Name:	News Server Request
Alias:	None
Where & How It is used:	NNTP Service Input
Description:	“Requests coming to nntp service”

Name:	NNTP Service Response
Alias:	None
Where & How It is used:	NNTP Service Output
Description:	“Response coming from nntp service”

Name:	NNTP Service Request
Alias:	None
Where & How It is used:	NNTP Service Output
Description:	“Requests coming from nntp service

Name:	File Engine Module Response
Alias:	None
Where & How It is used:	NNTP Service Input
Description:	“Responses coming to nntp service

Name:	NNTP Service Request
Alias:	None
Where & How It is used:	NNTP Service Output
Description:	“Requests coming from nntp service

Name:	Database Engine Module Response
Alias:	None
Where & How It is used:	NNTP Service Input
Description:	“Responses coming to nntp service

Name:	RSS Client Server Request
Alias:	None
Where & How It is used:	XML Service Input
Description:	“Requests coming to xml service”

Name:	XML Service Response
Alias:	None
Where & How It is used:	XML Service Output
Description:	“Response coming from xml service”

Name:	XML Service Request
Alias:	None
Where & How It is used:	XML Service Output
Description:	“Requests coming from xml service

Name:	File Engine Module Response
Alias:	None
Where & How It is used:	XML Service Input
Description:	“Responses coming to xml service

Name:	XML Service Request
Alias:	None
Where & How It is used:	XML Service Output
Description:	“Requests coming from xml service

Name:	Database Engine Module Response
Alias:	None
Where & How It is used:	XML Service Input
Description:	“Responses coming to xml service

Name:	Web Server Request
Alias:	None
Where & How It is used:	HTTP Service Input
Description:	“Requests coming to http service”

Name:	HTTP Service Response
Alias:	None
Where & How It is used:	HTTP Service Output
Description:	“Response coming from http service”

Name:	HTTP Service Request
Alias:	None
Where & How It is used:	HTTP Service Output
Description:	“Requests coming from http service

Name:	File Engine Module Response
Alias:	None
Where & How It is used:	HTTP Service Input
Description:	“Responses coming to http service

Name:	HTTP Service Request
Alias:	None
Where & How It is used:	HTTP Service Output
Description:	“Requests coming from http service

Name:	Database Engine Module Response
Alias:	None
Where & How It is used:	HTTP Service Input
Description:	“Responses coming to http service

4.2.2 Data Dictionary of Web Application

Name:	Registration info
Alias:	None
Where & How it is used:	Visitor Input
Description:	“written input data and interface commands by visitor in order to registration to the system”

Name:	User commands and data
Alias:	None
Where & How it is used:	User Input
Description:	“written input data and interface commands by user”

Name:	Admin commands and data
Alias:	None
Where & How it is used:	Administrator Input
Description:	“written input data and interface commands by admin”

Name:	Visitor commands and data
Alias:	None
Where & How it is used:	Visitor Input
Description:	“written input data and interface commands by user”

Name:	Visitor display info
Alias:	None
Where & How it is used:	Output of the *Newstreamline* News system
Description:	“output of the system which is shown on the visitor monitor”

Name:	User display info
Alias:	None
Where & How it is used:	Output of the *Newstreamline* News system
Description:	“output of the system which is shown on the user monitor”

Name:	Admin display info
Alias:	None
Where & How it is used:	Output of the *Newstreamline* News system
Description:	“output of the system which is shown on the admin monitor”

Name:	Request data
Alias:	Request
Where & How it is used:	Input of *Core* according to the *Newstreamline* system
Description:	“the input request data coming from user by web interaction”

Name:	Response data
Alias:	Response
Where & How it is used:	Output of the core system to the responding requests & Input of the *Newstreamline* system
Description:	“the output response data to the request data coming from core”

Name:	Login status
Alias:	None
Where & How it is used:	Output of *Newstreamline*(0.0)
Description:	“it is the desired input coming from visitor panel to register to the system”

Name:	Password & Id
Alias:	Visitor commands output data
Where & How it is used:	Output of the Registration(1.0)
Description:	"given password & id to the visitor, it is a display message of the Visitor Display Panel "

Name:	Personal info
Alias:	None
Where & How it is used:	Input of the Registration(1.0)
Description:	"The personal information of the visitor for registering to the system like name, surname, e-mail, etc."

Name:	Approval message
Alias:	None
Where & How it is used:	Output of Registration(0.0)
Description:	"It is an output to the visitor for after Registration(1.0) process has been completed"

Name:	Password & ID
Alias:	Id & Password
Where & How it is used:	Input of User Interaction(2.0)
Description:	"the id and password of user/admin given by the system in order to login"

Name:	Invalid data
Alias:	None
Where & How it is used:	Output of the User Interaction(2.0)
Description:	"an error message that shows the invalidation of Id and password, it does not let the user/admin log in"

Name:	Interaction Request
Alias:	Request data
Where & How it is used:	Input of *Core* according to the User Interaction(2.0)
Description:	“It is an request for being a member of the system”

Name:	Interaction Response
Alias:	Response data
Where & How it is used:	Output of *Core*
Description:	“it is a response to the system for logging in to the system”

Name:	Read / Write Post Request
Alias:	None
Where & How it is used:	Input of User News Operations(3.0)
Description:	“the request for reading or writing a post to the system”

Name:	Send Post Request
Alias:	None
Where & How it is used:	Input of User News Operations(3.0)
Description:	“the request for sending a post to the system”

Name:	Search Post Request
Alias:	None
Where & How it is used:	Input of User News Operations(3.0)
Description:	“the request for searching a post of the system”

Name:	Operation Request
Alias:	Request data
Where & How it is used:	Input of *Core* according to the News Operations(3.0)
Description:	“it is an request for doing news operations”

Name:	Operation Response
Alias:	Response data
Where & How it is used:	Output of *Core*
Description:	“it is a response to the system doing news operations”

Name:	Output response
Alias:	Output data
Where & How it is used:	Output of User News Operations(3.0)
Description:	“it is a response to the system for logging in to the system”

Name:	Modification data
Alias:	None
Where & How it is used:	Input to the Personal info modification(4.0)
Description:	“it is necessary data to modify personal information”

Name:	Modification Request
Alias:	Request data
Where & How it is used:	Input of *Core* according to the Personal info modification(4.0)
Description:	“it is an request for updating personal information”

Name:	Modification Response
Alias:	Response data
Where & How it is used:	Output of *Core*
Description:	“it is a response to the system for updating personal information”

Name:	Modified user data
Alias:	User display info
Where & How it is used:	Output of the Personal info modification(4.0)
Description:	“it is a display message according to the modification of the user data”

Name:	Modified admin data
Alias:	Admin display info
Where & How it is used:	Output of the Personal info modification(4.0)
Description:	"it is a display message according to the modification of the admin data"

Name:	Configure news groups
Alias:	None
Where & How it is used:	Input of the Newsserver configuration(5.0)
Description:	"it is necessary data to do configuration about news groups"

Name:	Configure member info
Alias:	None
Where & How it is used:	Input of the Newsserver configuration(5.0)
Description:	"it is necessary data to do configuration member information"

Name:	Configuration Request
Alias:	Request data
Where & How it is used:	Input of *Core* according to the Newsserver configuration(5.0)
Description:	"it is an request for configuring the system"

Name:	Configuration Response
Alias:	Response data
Where & How it is used:	Output of *Core*
Description:	"it is a response to the system for configuring the system"

Name:	Configured news group info
Alias:	Admin display info
Where & How it is used:	Output of the Newsserver configuration(5.0)
Description:	"it is an admin display message according to the configuration of the news group information"

Name:	Configured member info
Alias:	Admin display info
Where & How it is used:	Output of the Newsserver configuration(5.0)
Description:	"it is an admin display message according to the configuration of the members' information"

5. SYSTEM ARCHITECTURE

5.1 System Modules

5.1.1 News Service Module

Functionality of a News Service module is divided into two: communication with a News Server Client, and communication with a File Query Engine.

Communication with a News Server Client includes several functions but the main thing that module does in this part is continuous listening to the port which news server connected to. If any request comes, it processes it and sends that request to the File Query Engine. If information from file system is wanted, same mechanism works in inverse directions; information from file system comes to module through File Query Engine and passed to News Server Client.

Requests that come from news service client are:

- Start/Stop Service
- Read Post
- Write Post
- Send Post

Requests that are send to File Query System

- Update Post Directory
- Select Post from Post Directory
- Create Post Directory

NewsServerClient connects to NewsServer which is in NewsService Module and starts to interact with it. Here NewsServerClient represents a user (client) of the News Server that send different requests to NewsService. NewsService process that request and forwards them to File Query Engine. There are three functionalities of NewsService.

It creates data class ,Post that will be used by File Query Engine and passes this class to it. File Query Engine takes the income data and (do xml parsing and) generates file queries. Then, it connects to File System and stores data to File System.

It passes the PostId or GroupId to File Query Engine, which connects to File System and retrieves the appropriate information according to given data. Retrieved data sent back to NewsService and to NewsServerClient through it.

5.1.2 Web Service Module

For web applications, we use web application GUI class. It depends on several classes namely PersonalInfoScreen, ResultScreen, Loginscreen, SearchScreen, PostScreen and AdminInterface. When system starts there exist severeal objecs of these classes.
According to user type display screen will change. For example, admin type user can modify system capabilities so there exists a adminInterface class.

In order to choose which kind of user trying to enter to the system, there exists a loginScreen class. After login, a regular user may want to change his or her personal information. For this purpose, there exists a PersonalInformation class. With the help of this class, user can easily modify his/her own settings. Moreover, user indicates his/her mail preference to the system from this dialog. What I mean from the mail preference is that, some users may want to follow some newsgroups from mail accounts. Therefore, when a new article has posted to a newsgroup, our system sends mail to user.
User can search in posts according to subject and title. SearchScreen class is created for this purpose. Result screen class is used for displaying results of user requests.
Webservice class is an important part of our system since it is husnas part. All user operations can be done via this class. Webservice class calls the WebApplicationGUI to handle user requests. After handling requests from GUI, Webservice class invokes the XML/SQLEngine class and File Query Engine. XML/SQLEngine class can talk with database. This class has a parser class to select which type of request is formed. Then, it creates a request object which

is used by database. On the other hand , File Query Engine class can talk with file system and it can create a request object which is used by file system

Request object types are searchRequest, StatisticsRequest, ModificationRequest, RSSRequest and IdentificationRequest.

In our system, there are two places to save information. For posts data, we have constructed a file system. For user information and their relations with news groups are saved in database. Consequently, we have used File Query Engine for some user requests, like read post, write post, search post, etc. SQL/XML Engine is used for other user requests like create news group, register to system, login, rss feed, mail etc.

Database will execute the request and creates a result object. This result object reaches ResultScreen through firstly XML/SQL Engine and then WebService.

5.1.3 Mail Service Module

MailServerClient connects to MailServer which via Web Service. The client send request to Web Service, in order to get mail from subscribed news groups. Web Service acts as a transporter and forward it to SQL/XML Service and File Query Service. In other words, when a user wants to get mail from a specific news group, he/she select this group. Then this information is saved in Mail_List Table in database. When a news is posted, a mail list is constructed. Then, Mail Service Module, send this post according to mail list.

5.1.4 SQL/XML Engine Module

SQL/XML Engine listens Mail Service, Web Service in the core. It constructs queries with respect to incoming requests from services. It puts these queries to the database in order. These queries are executed. The result set of executed queries are sent to related destinations. Moreover, database sends all triggers and data via SQL/XML Engine.

5.1.5 File Query Engine Module

Since news service save the posts to file system, we ave added this module to our project to make connection between services and file system. File Query Engine listens News Service, Web Service, and Mail Service. It constructs queries with respect to incoming requests from services. These requests have to be related with post operations. It puts these queries to the file system in order. These queries are executed. The result set of executed queries are sent to related destinations.

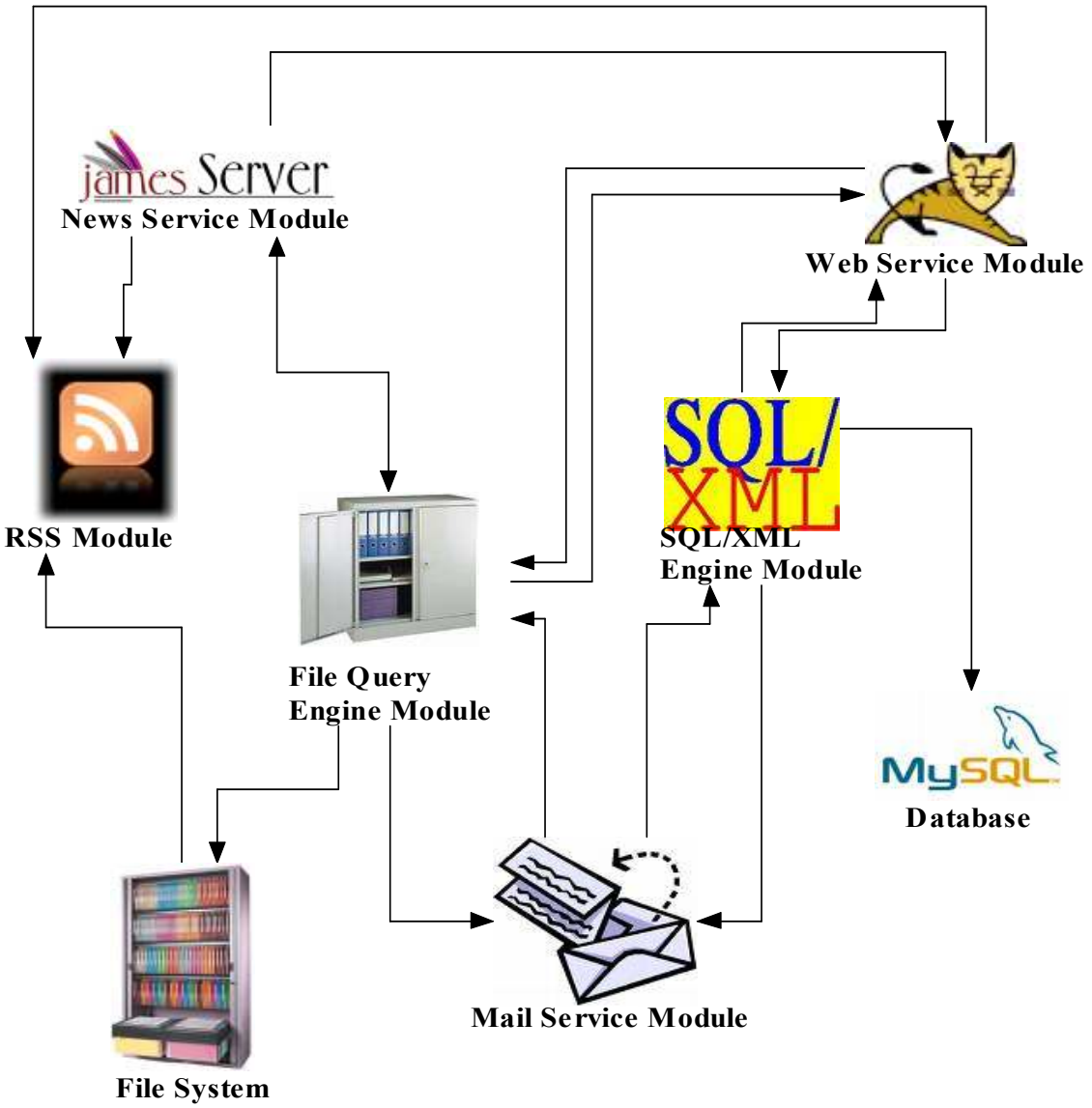
5.1.6 RSS Module

This service is responsible for constructing xml files according to RSS format. It constructs these files for each news group separately by always listening to file system for incoming news. If a new post is come RSS Module updates the related xml file. When a user wants to get an RSS news feed, the following steps has to be done:

1. He/she must have an RSS reader, in order to read.
2. He/she must write down the RSS links, which are present in Web Page.

Then, when news comes to the system, he/she can learn them from RSS reader without the opening web page obligation.

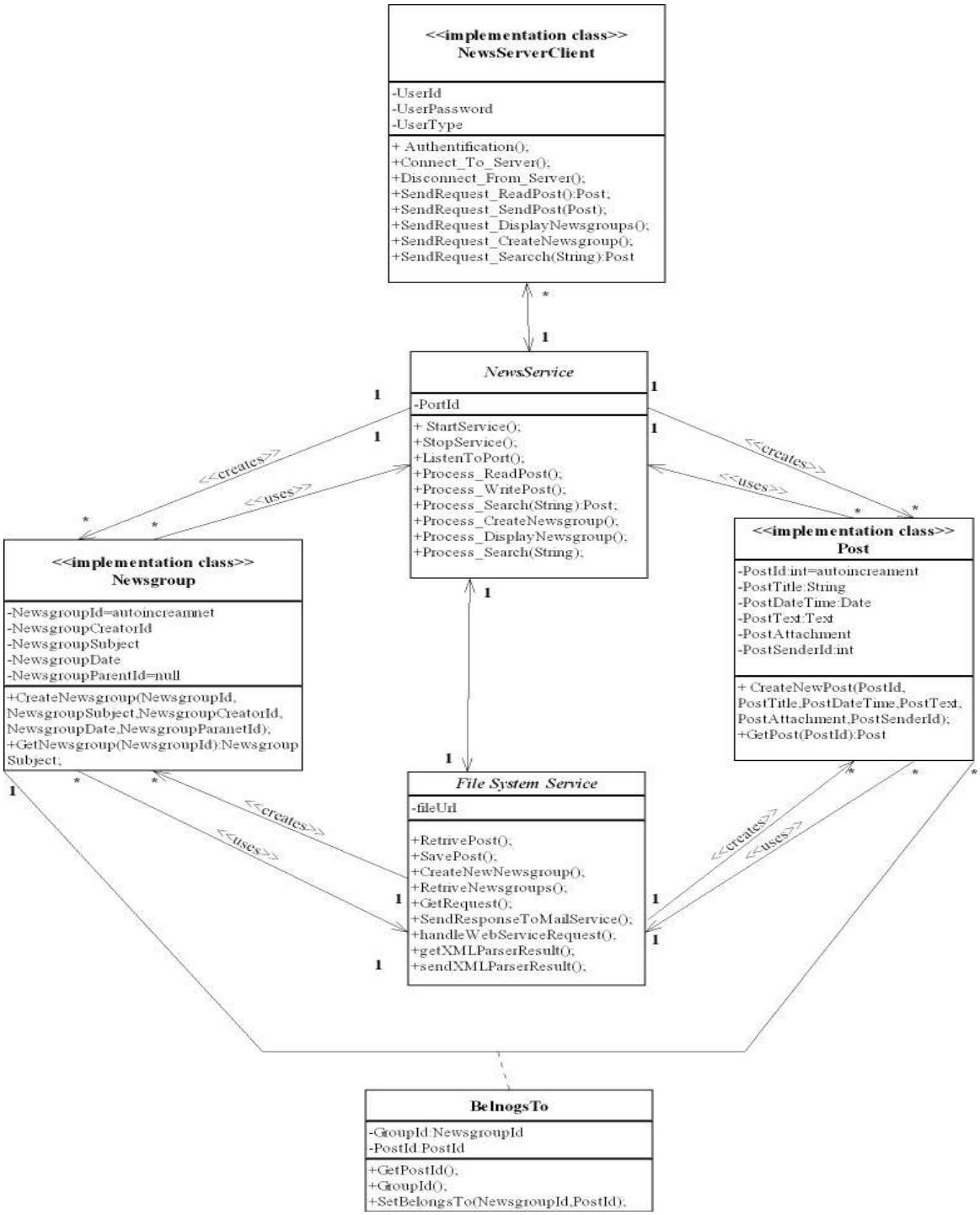
5.2 Architecture Diagram



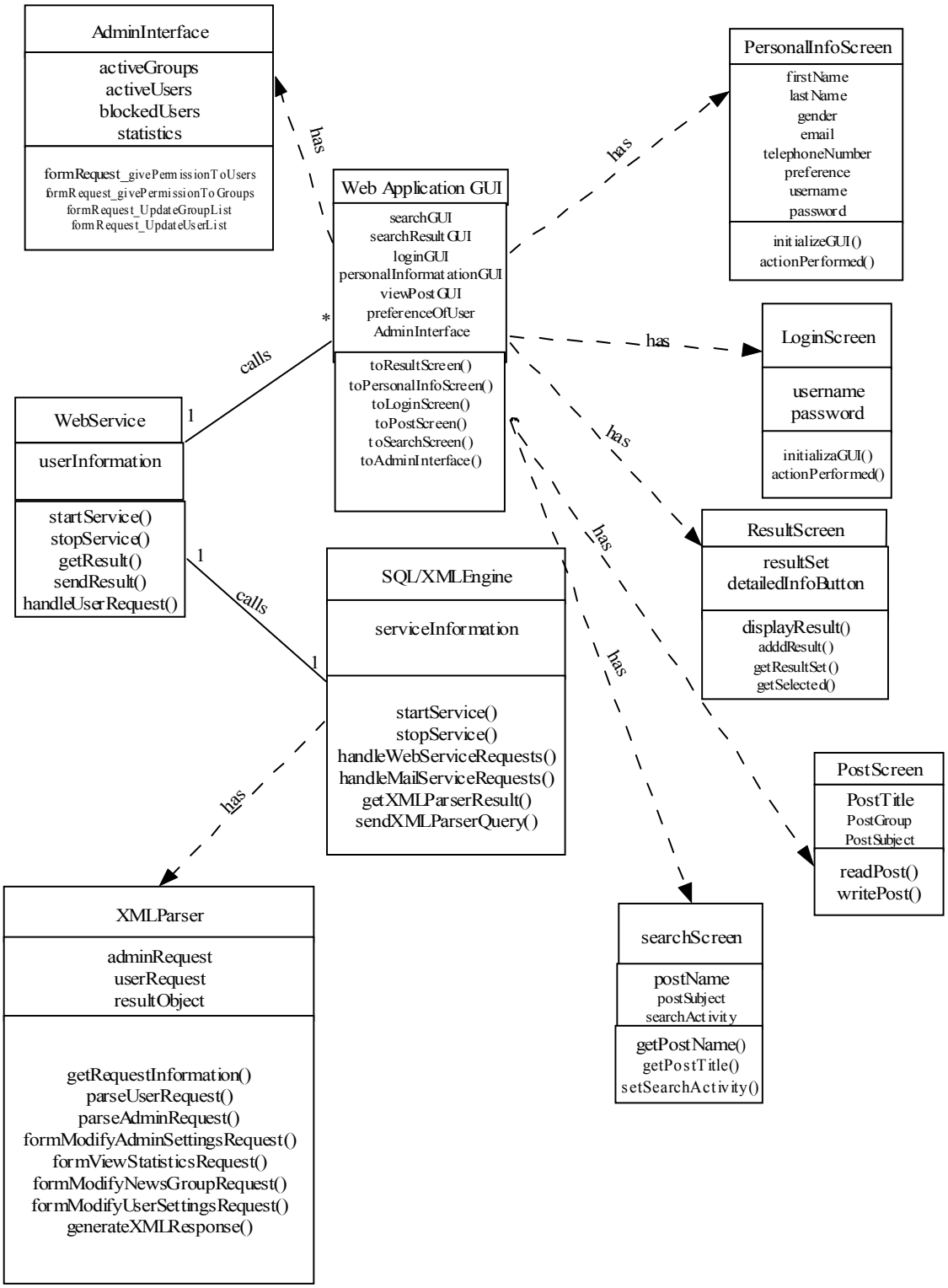
6. UML DIAGRAMS

6.1 Class Diagrams

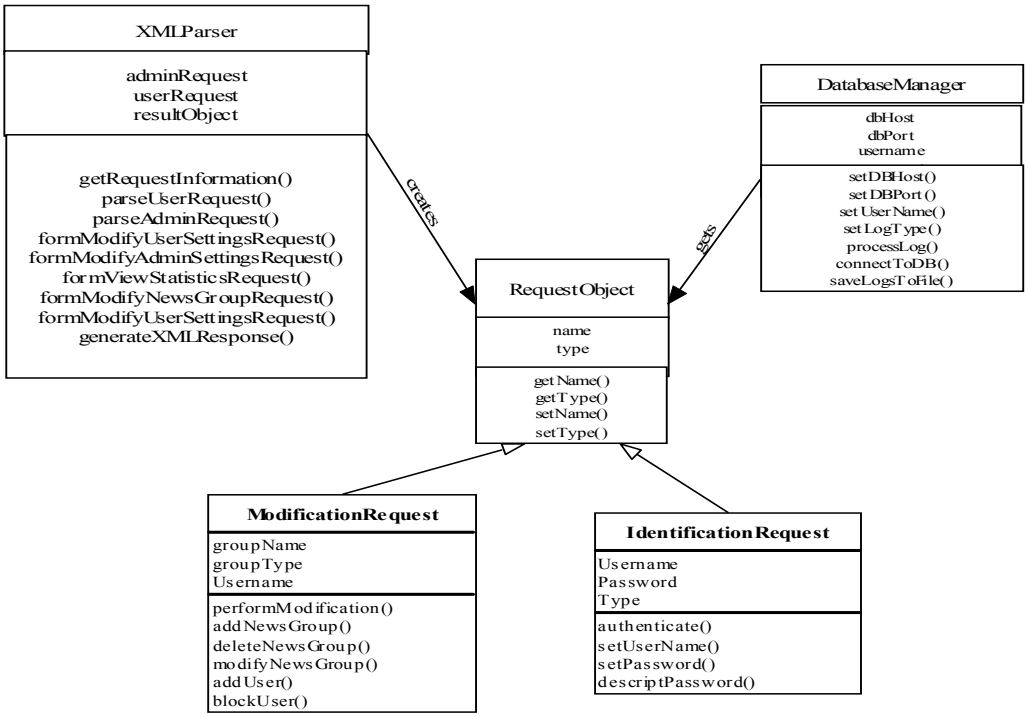
6.1.1 Class Diagram of News Service



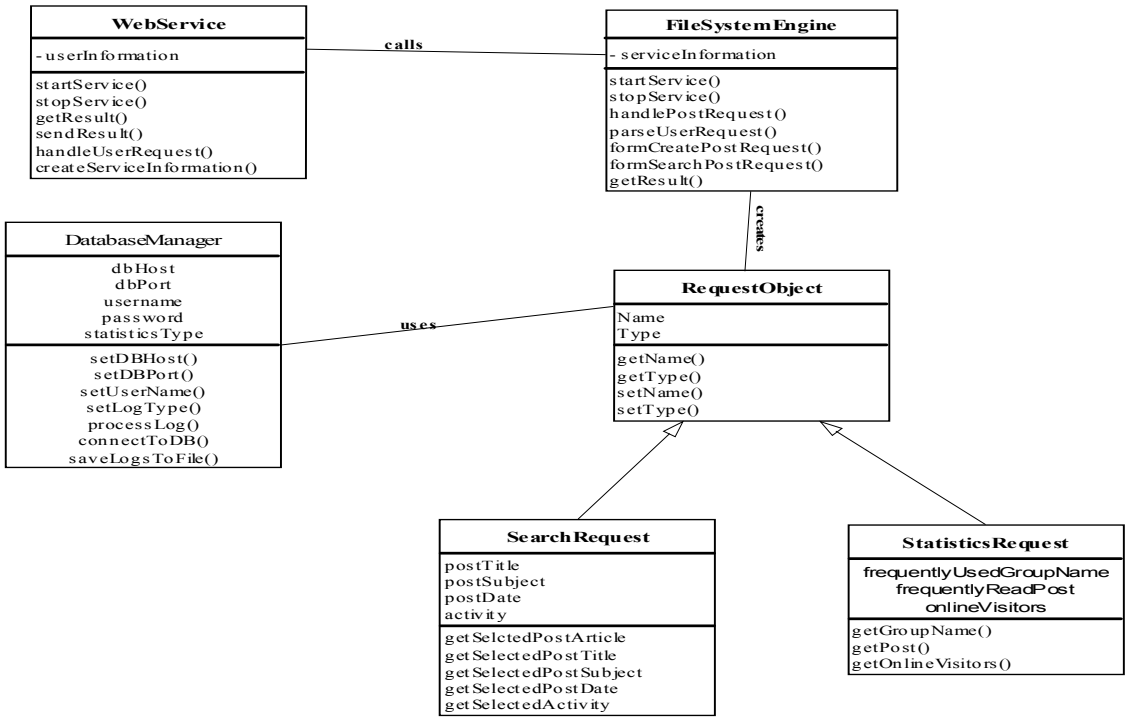
6.1.2 Class Diagrams of Web Service



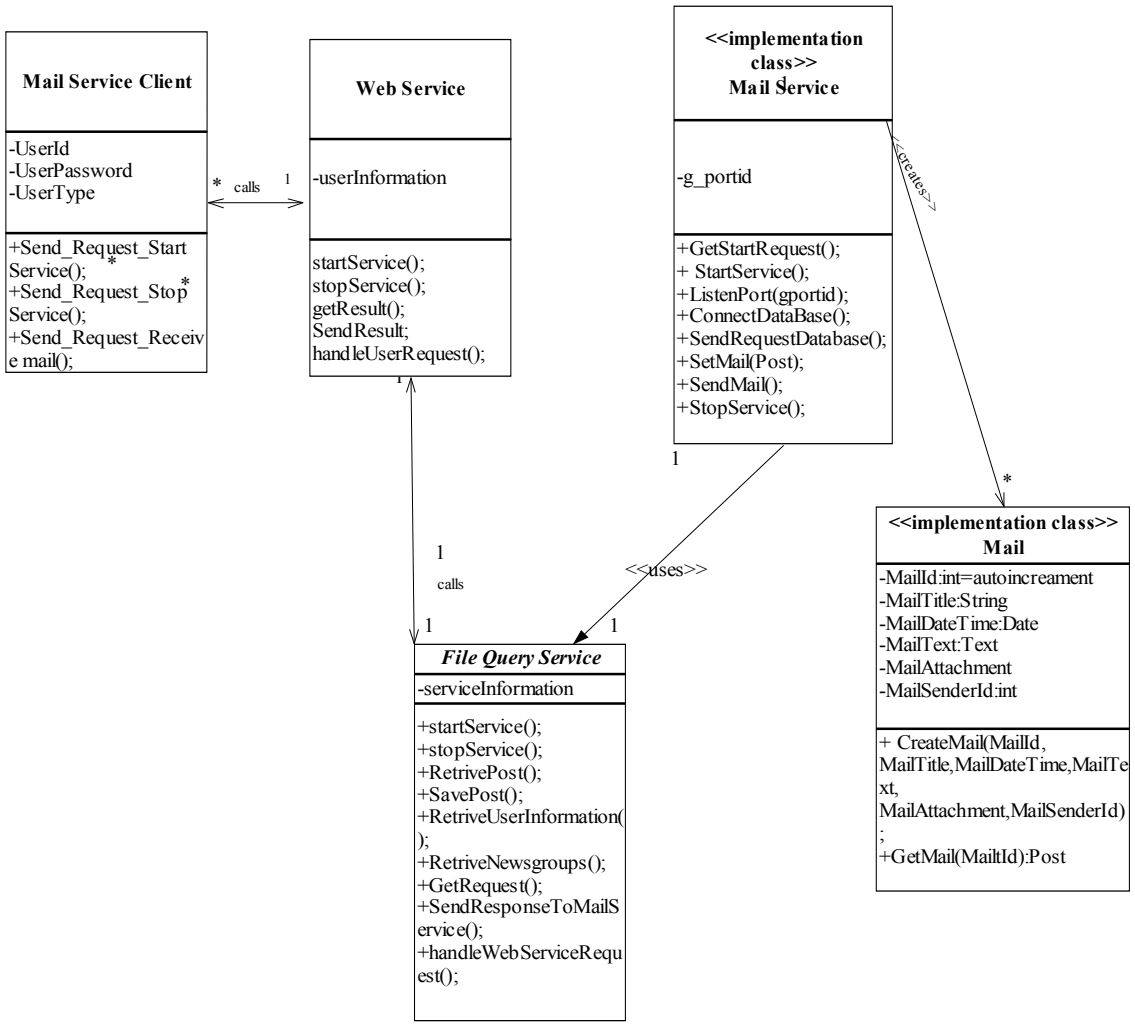
6.1.3 Class Diagram of Request Object - Database



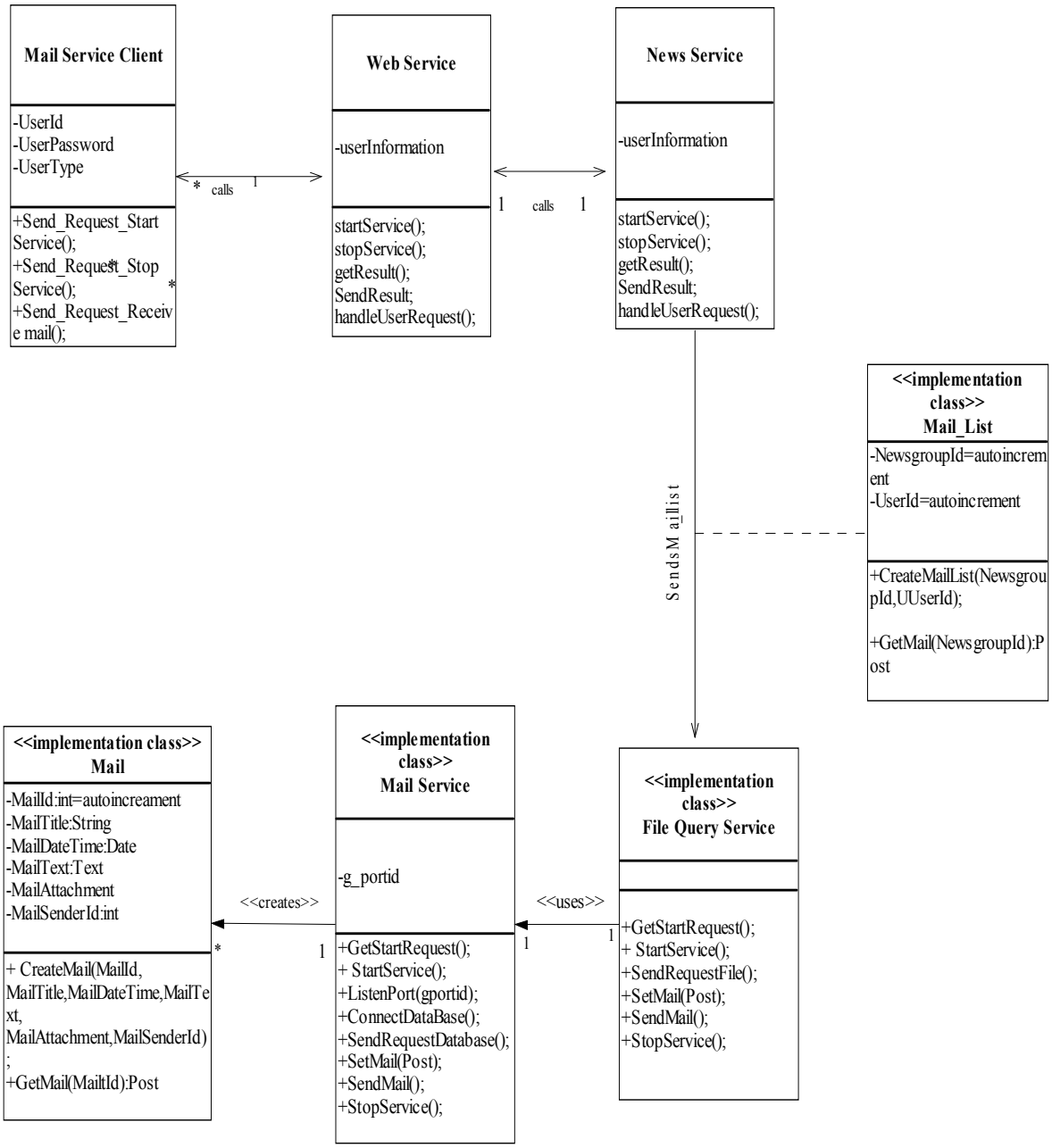
6.1.4 Class Diagram of Request Object – File System



6.1.5 Class Diagram of Mail Service

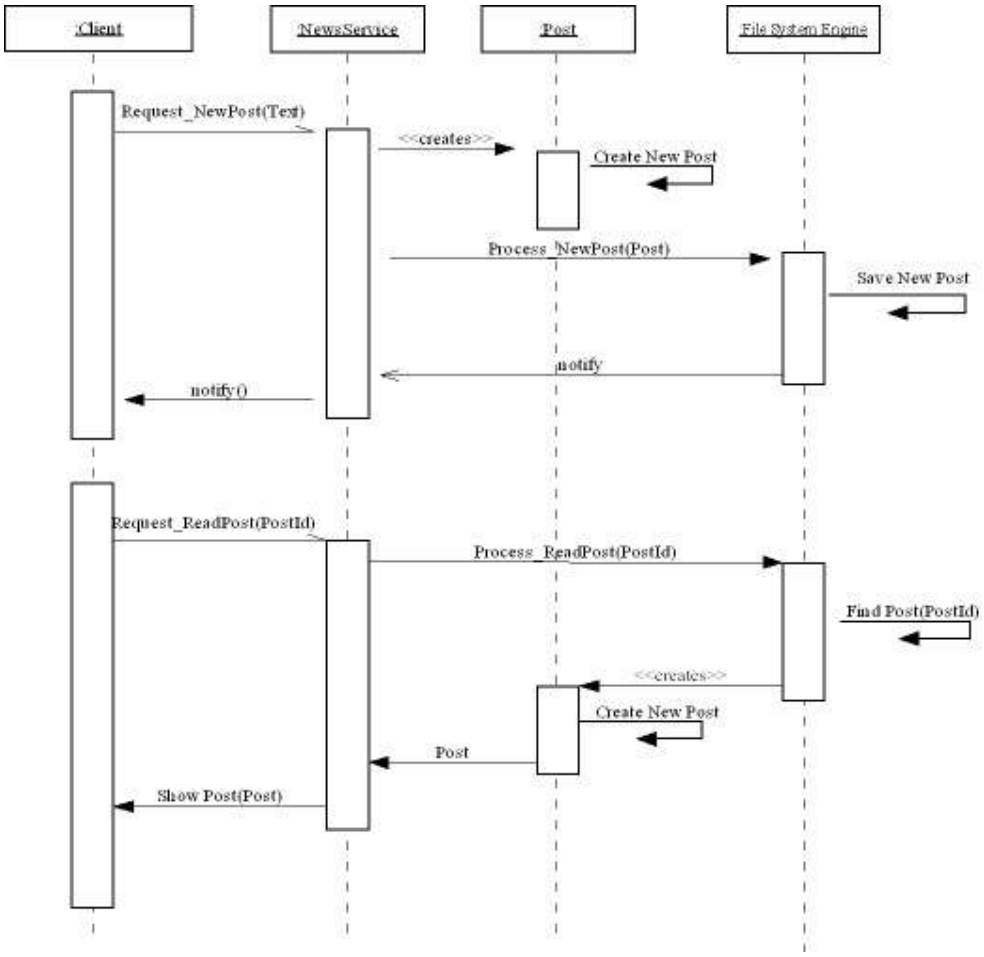


6.1.6 Class Diagram of Mail List



6.2 Sequence Diagrams

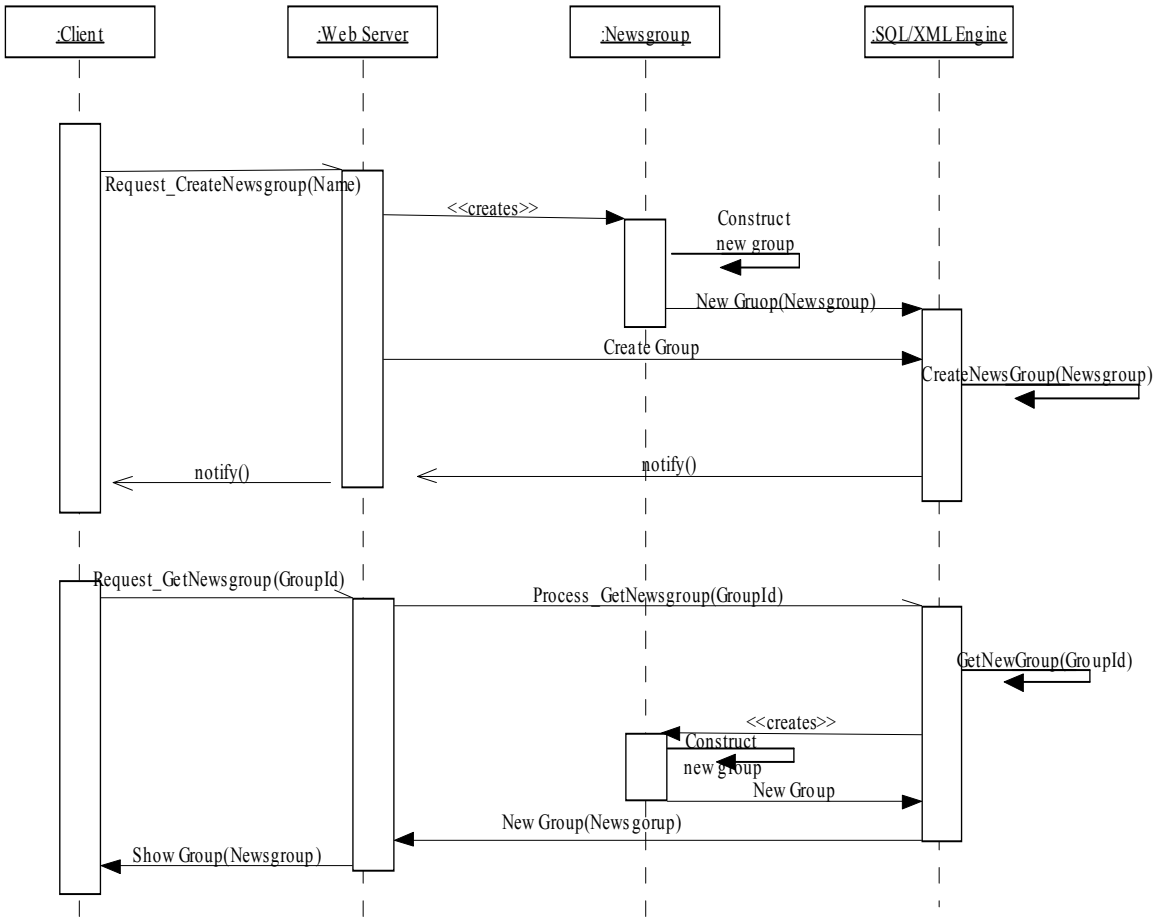
6.2.1 Sequence Diagram for Post



Client send request to the News Server to create a new Post by sending post data. News Server creates a new Post and passes it to the File System which saves that newly formed post. After post is saved, notification about the fulfillment of the request is forwarded to the client.

In the same way, client can send a request to view a particularly post. This time a post id is passed to the News Server that forwards it to the File System. If post is found, template for a post is created with the data from the File System and that newly created post is passed to the user through the News Server.

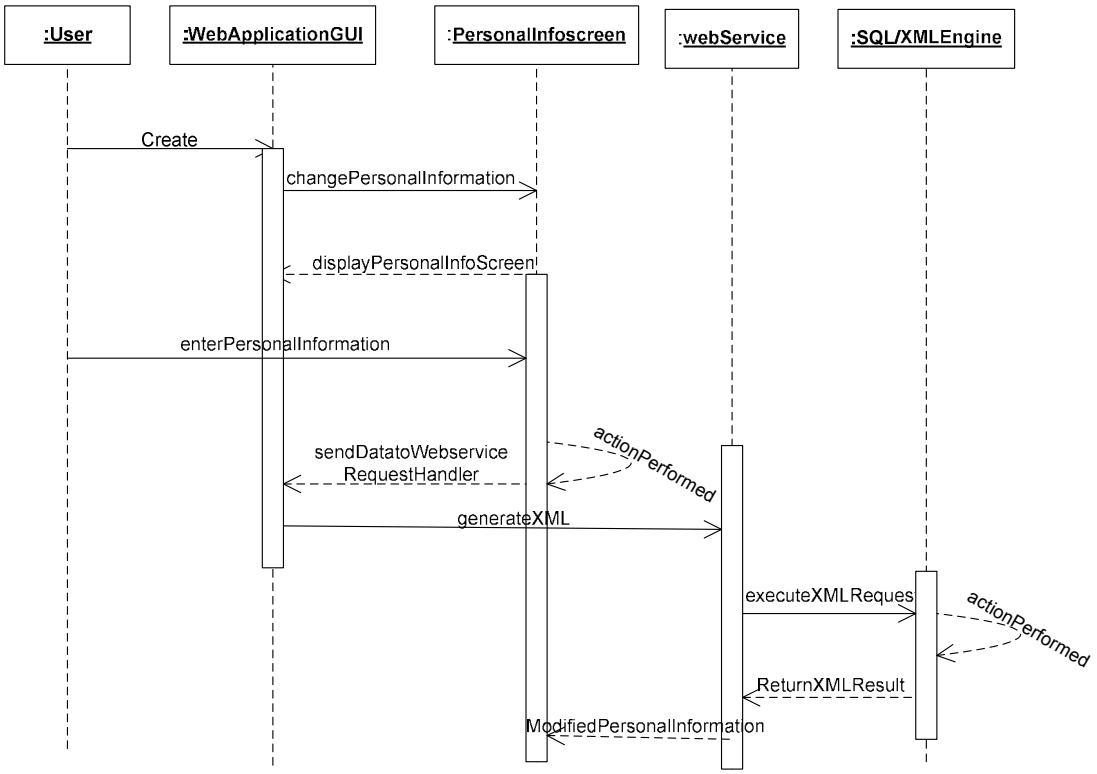
6.2.2 Sequence Diagram for Newsgroups



Administrator of the system send request to the Wes Server to create a new news group by sending name data. Web Server creates a new News Group and passes it to the database by SQL/XML engine. After news group is created, notification about the fulfillment of the request is forwarded to the administrator.

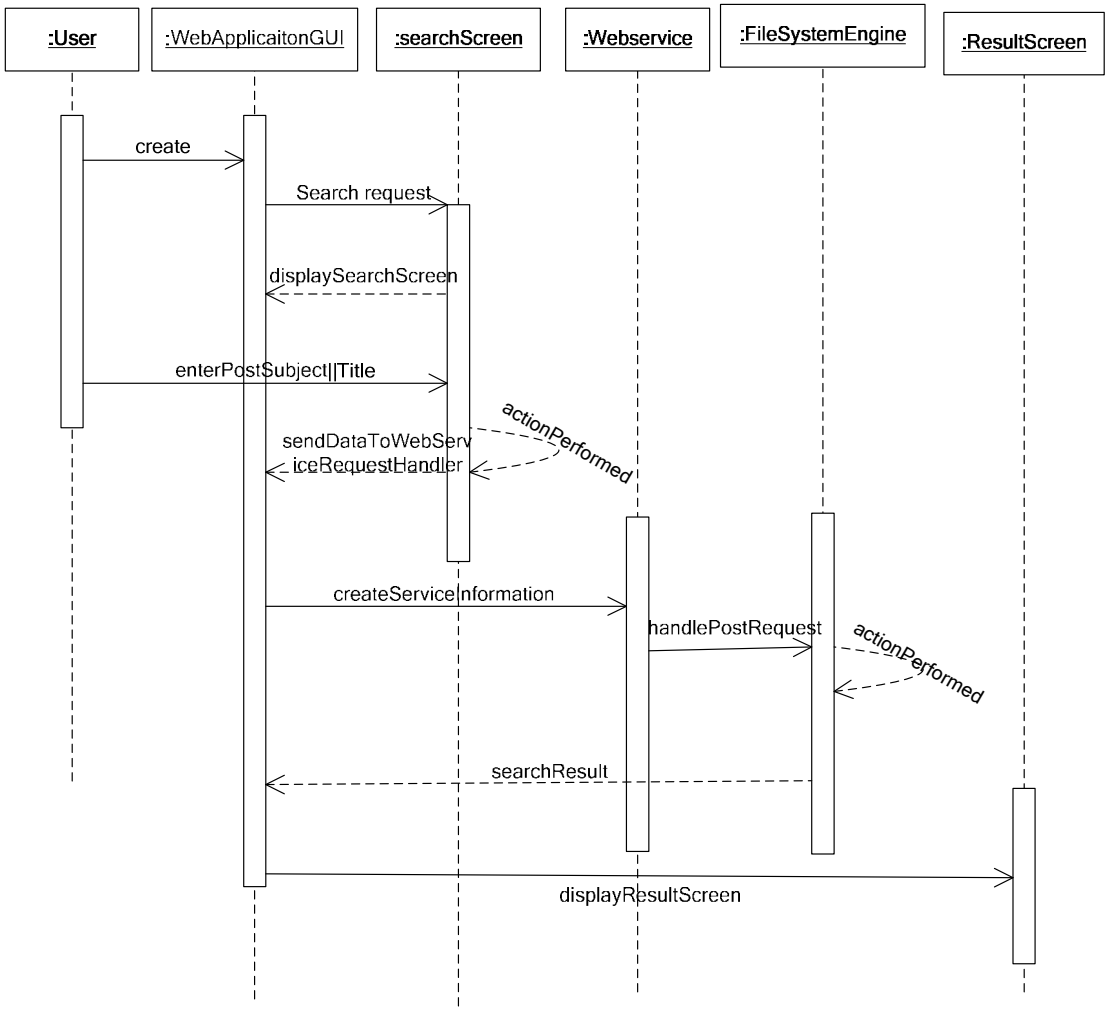
In the same way, administrator can send a request to view a particularly news group. This time a news group id is passed to the Web Server that forwards it to the SQL/XML Engine. If news group is found in database, it is shown in the administrator web page by Web Server.

6.2.3 Sequence Diagram for Update Personal Information



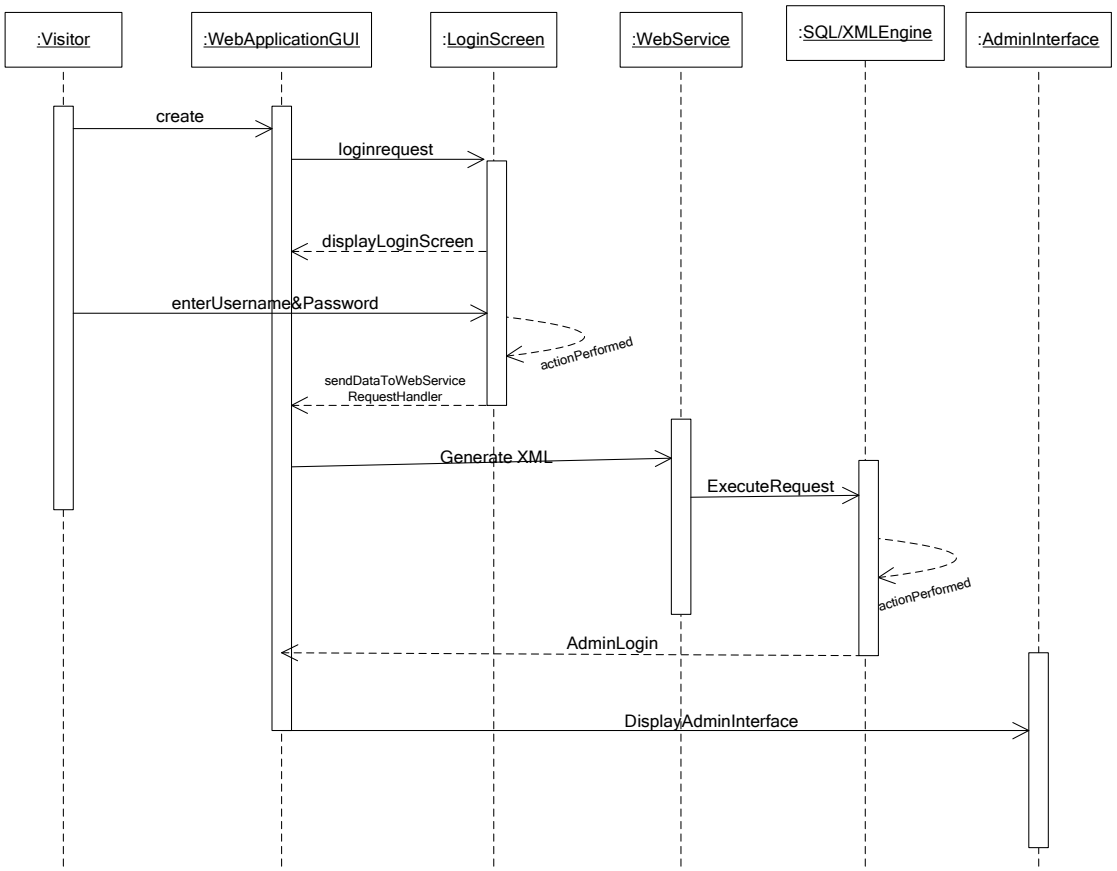
User can modify his/her personal information via Web Application GUI. He/she selects change personal information option from GUI. Then a personal information screen is opened that shows his/her personal information. User enters necessary data to update. After submission of data, Web Service sends request to SQL/XML Engine to perform update in database. When user personal information is updated in user table, by the help of SQL/XML Engine the updated personal data is shown via Web Service.

6.2.4 Sequence Diagram for Search Post



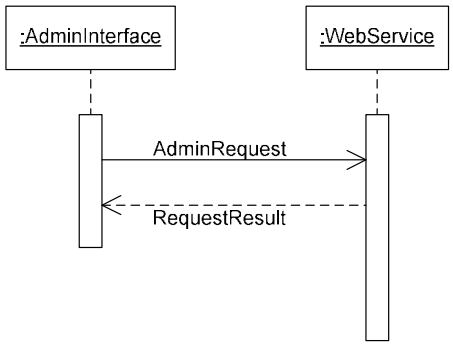
User can search a particular post via Web Application GUI. He/she selects search a post option from GUI. Then a search screen is opened. User enters either post subject or post title information. After submission of data, Web Service sends request to File Query Engine to perform search in posts which are recorded in file system. The result of search operation is shown in Result screen, via File System Engine firstly and via Web Service secondly.

6.2.5 Sequence Diagram for Admin Login



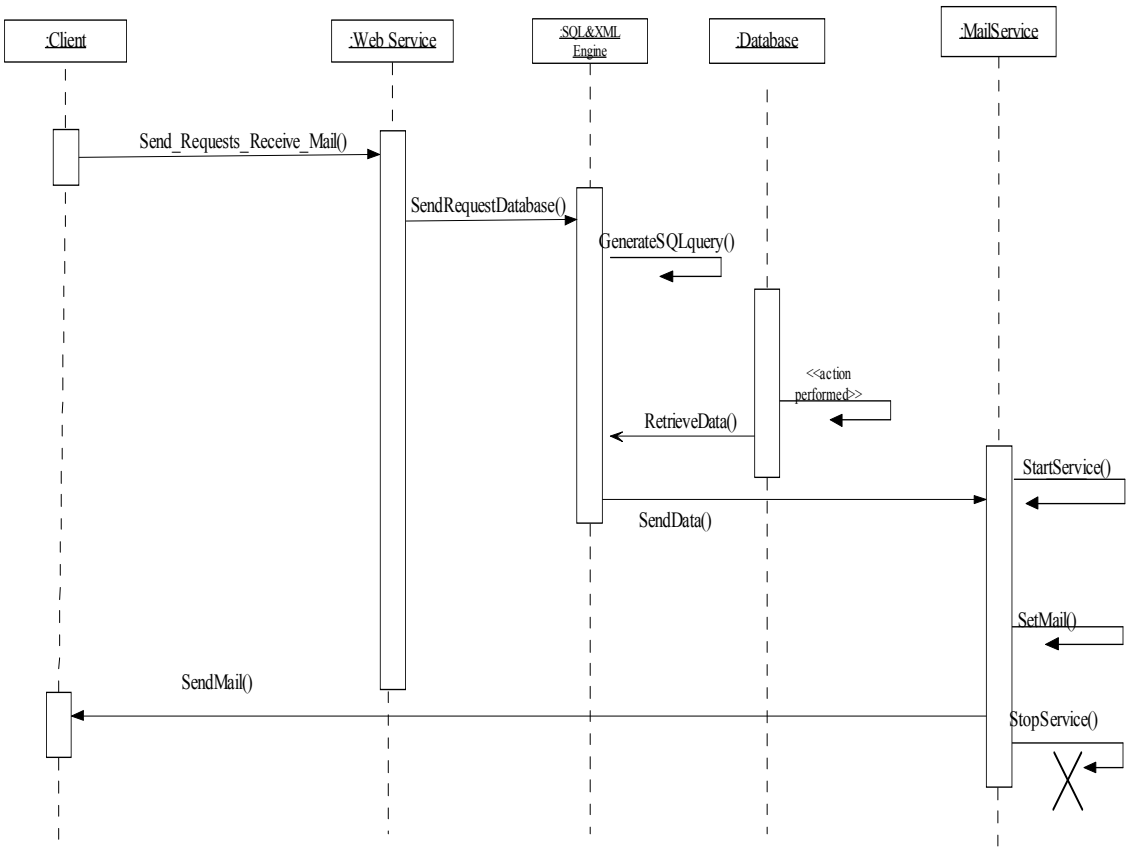
In admin login, actions performed are similar to the actions explained above. Admin enters username and password to web application GUI, and sends request to SQL/XML Engine via Web Service. SQL/XML Service executes query in user table and sends the result to GUI. If login is succeeded, then admin user interface will be opened.

6.2.6 Sequence Diagram of Admin Operations



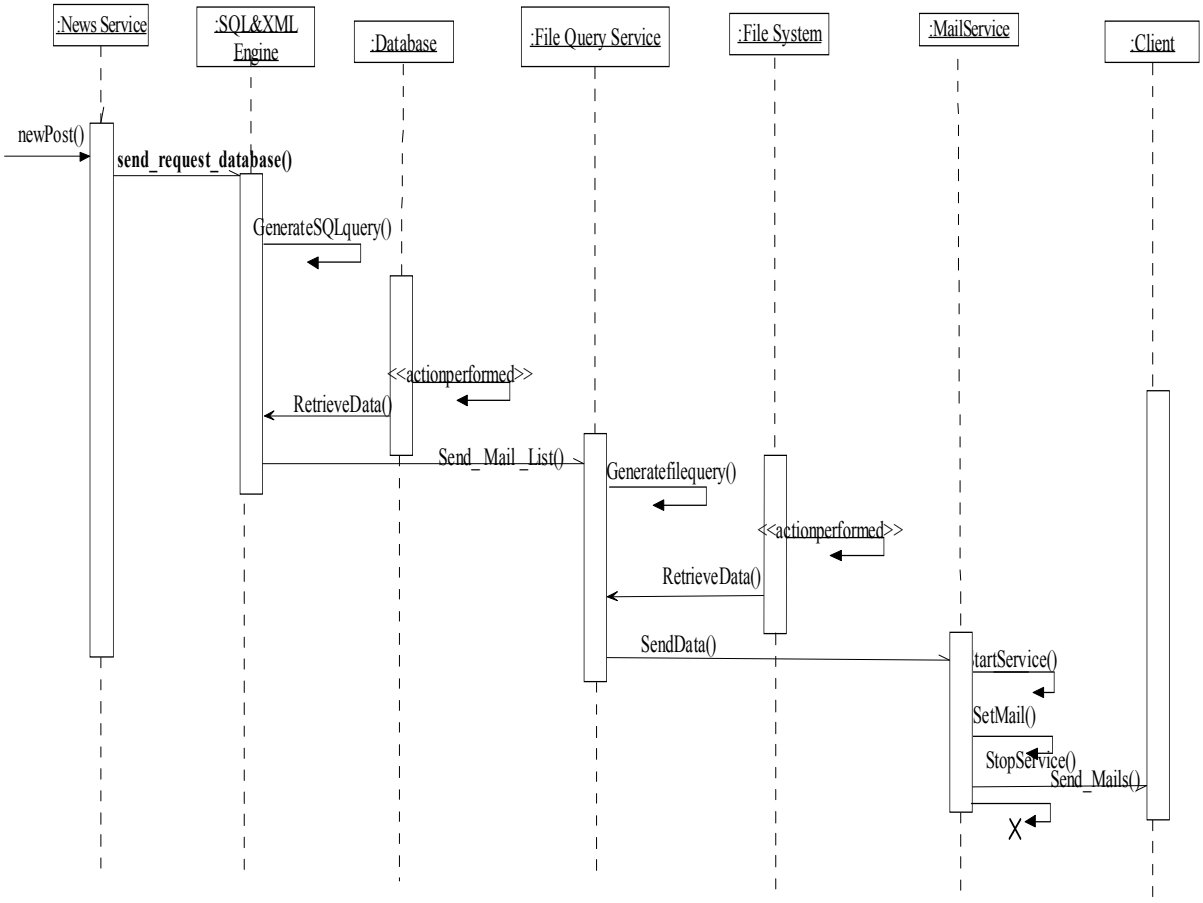
Admin sends all of requests to web service which connects to file system and database via File Query Engine and SQL/XML Engine. The responses to the resulted requests are sent to admin interface via Web Service again.

6.2.7 Sequence Diagram for Mail Service



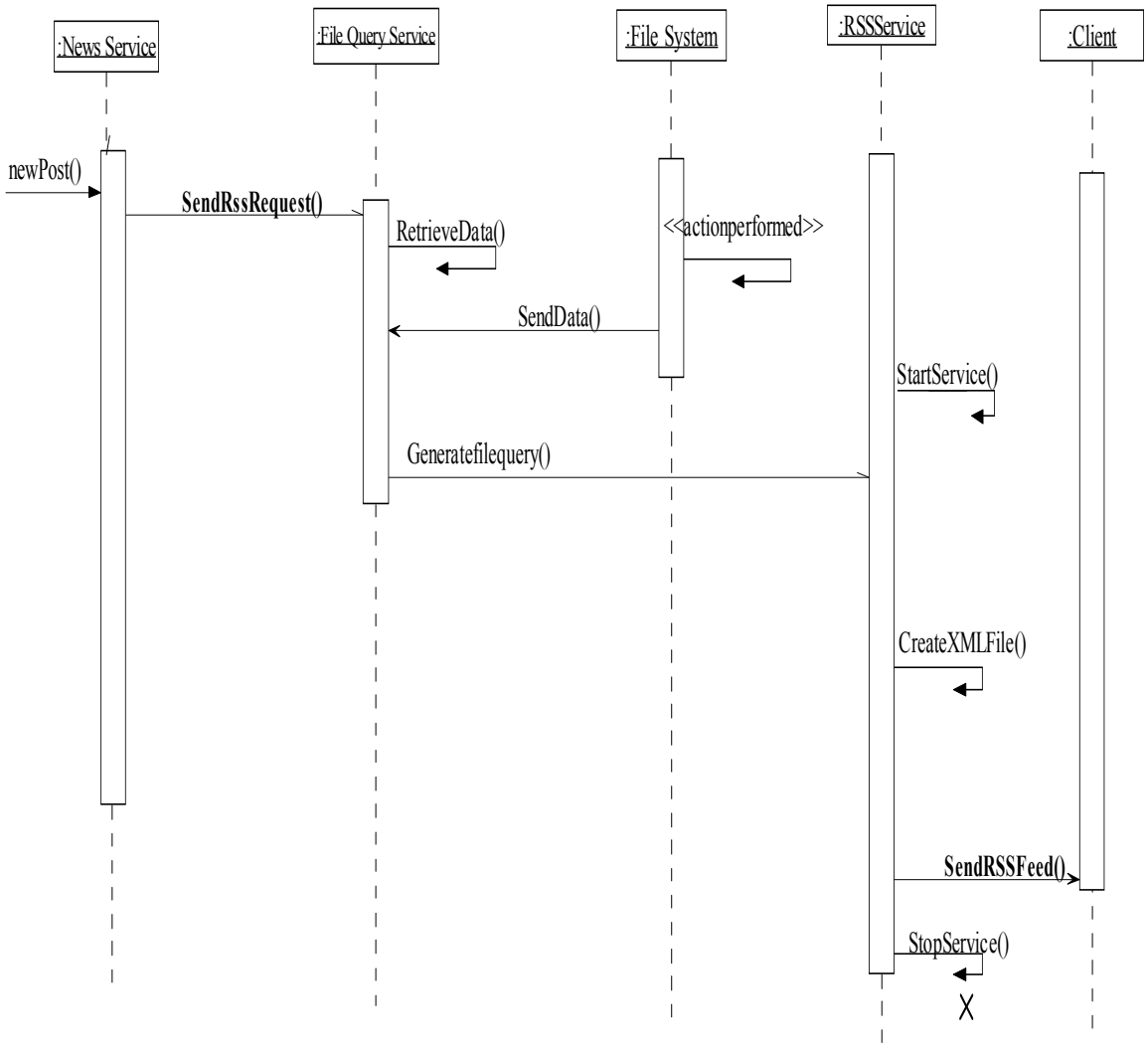
Client sends request to Web Service, if he/she wants to get an e-mail from system when a new post is come to the news group that he/she is subscribed to.

6.2.8 Sequence for mail list



When a new post is come and saved in file system, a request is send to SQL/XML Engine to create a mail list from database Mail_List table. After getting mail list from database, the post data is obtained from File System via File Query Service. Then both data are sent to the Mail Service, in order to make it send mail. Finally, mail including new post data is sent to clients in mail list.

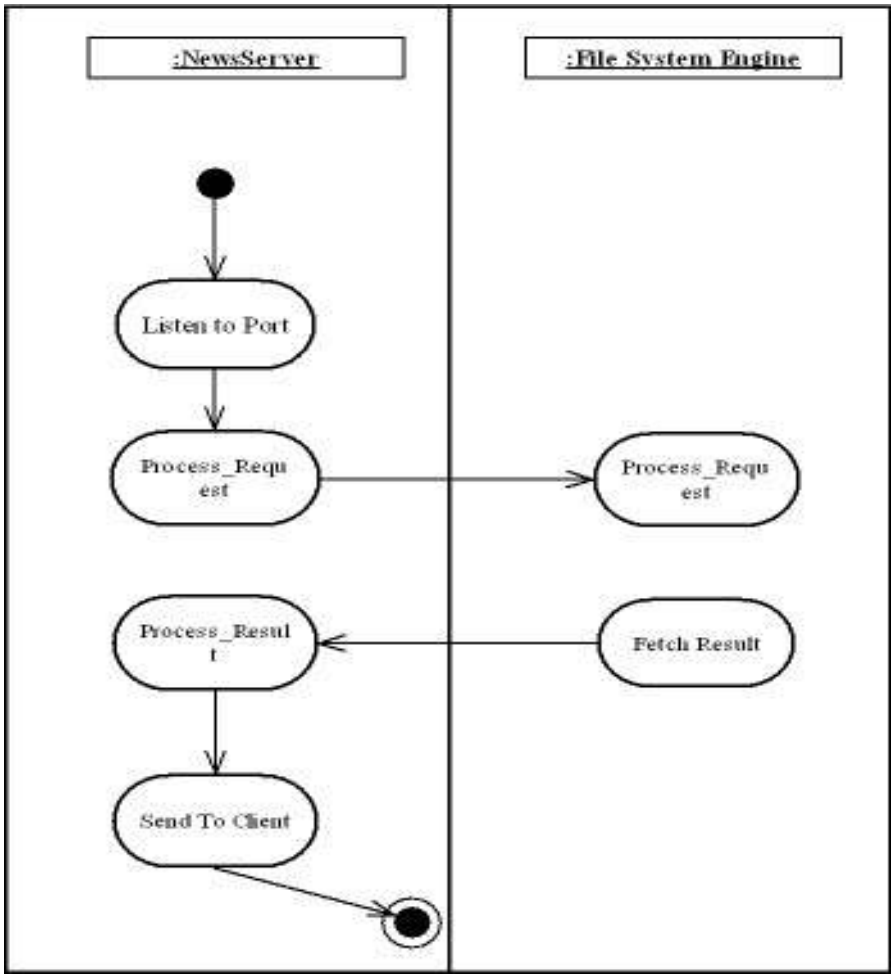
6.2.9 Sequence Diagram for RSS Request



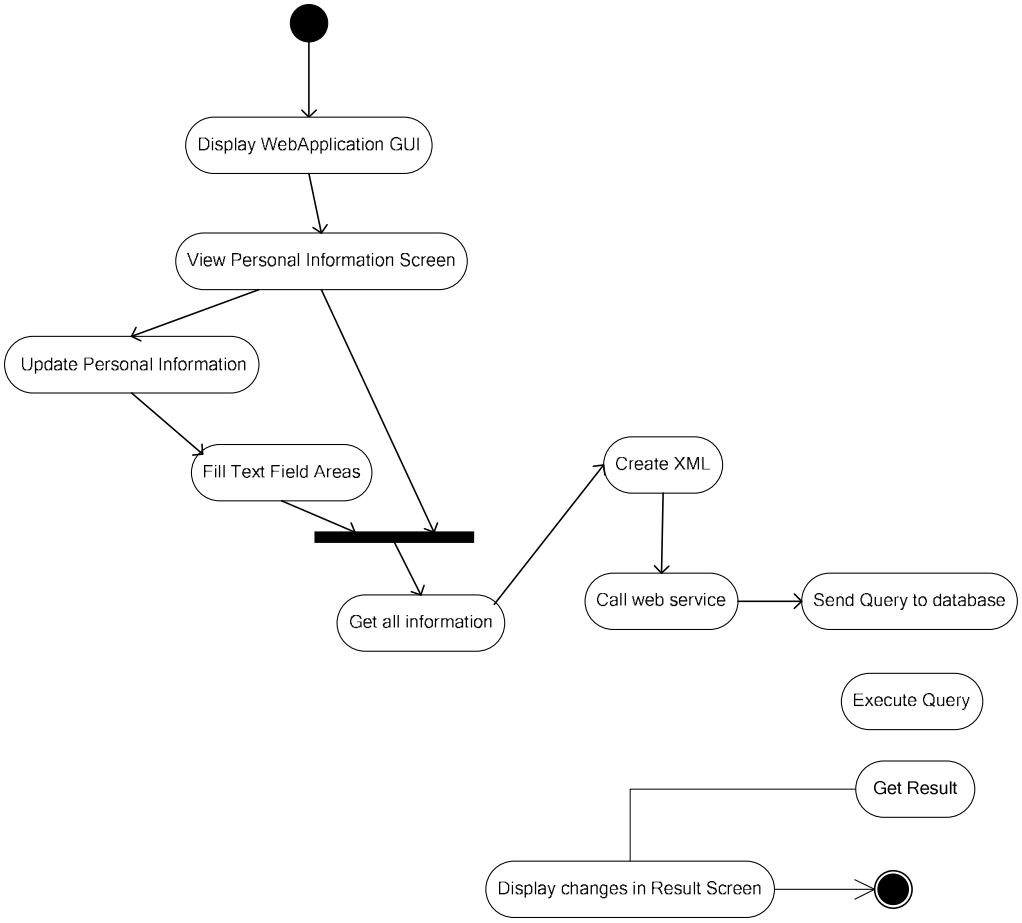
When a new post is come and saved in file system, a request is send File Query Service to obtain new post data from file system. Then, a request is sent to RSS module to make it generate an XML file in RSS format. The RSS news feed is obtained by client via RSS news feed reader which uses XML file generated by RSS Module.

6.3 Activity Diagrams

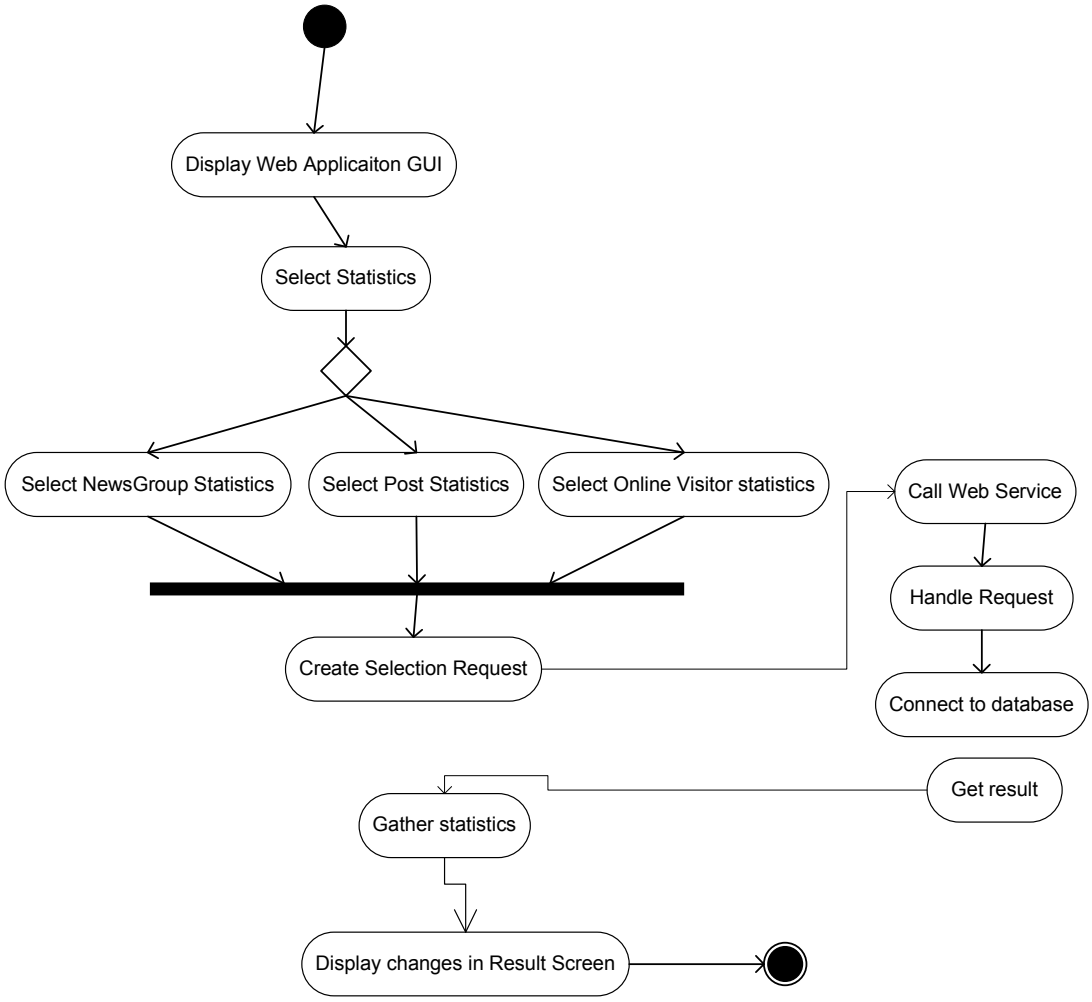
6.3.1 News Service Activity Diagram



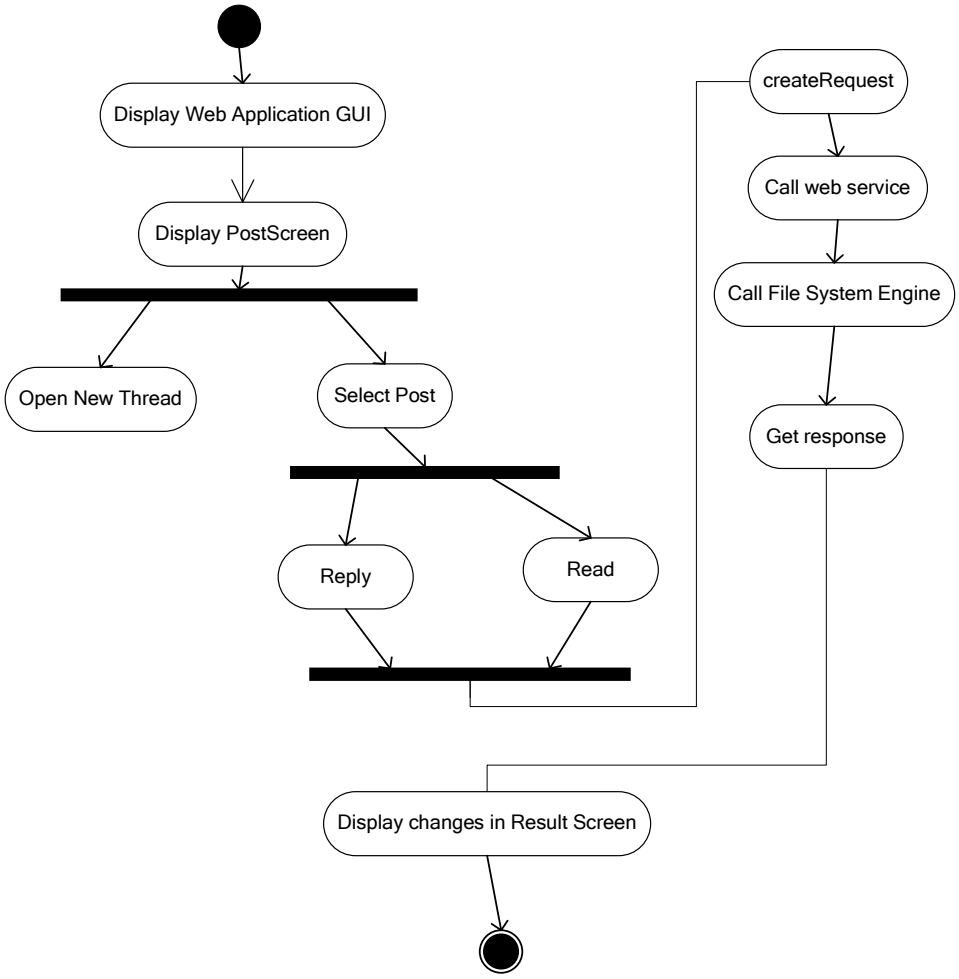
6.3.2 Update Personal Information Activity Diagram



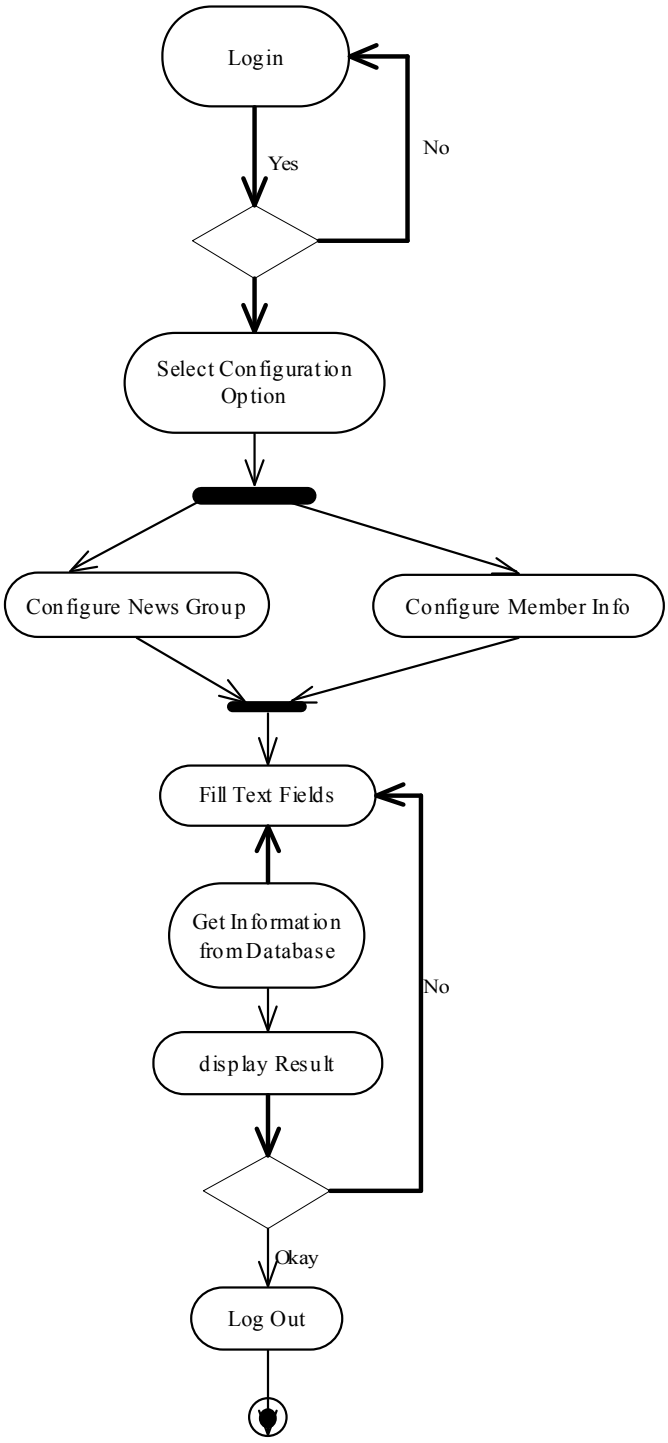
6.3.3 Request Statistics Activity Diagram



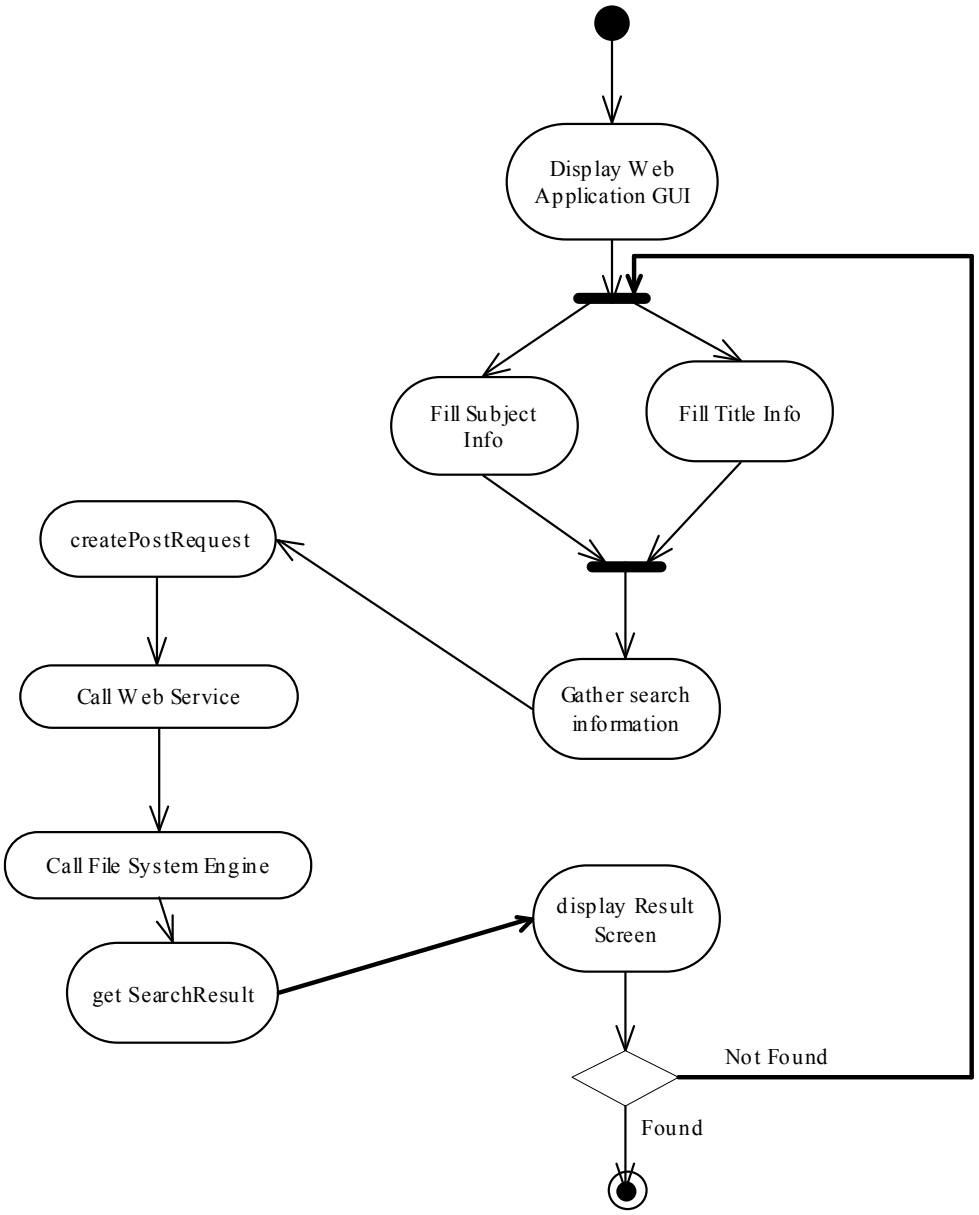
6.3.4 Post Related Requests Activity Diagram



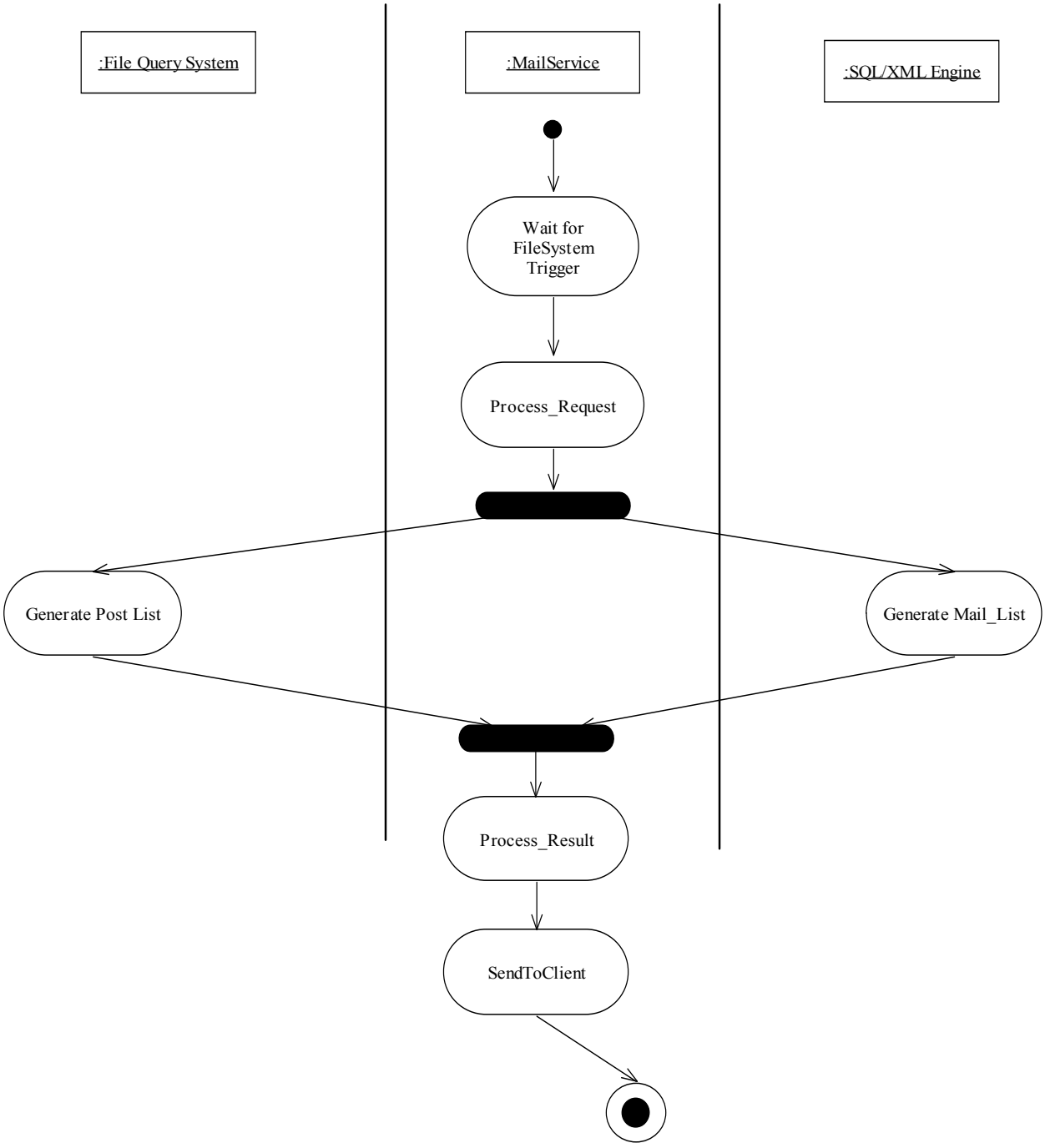
6.3.5 Activity Diagram of Admin User Interface



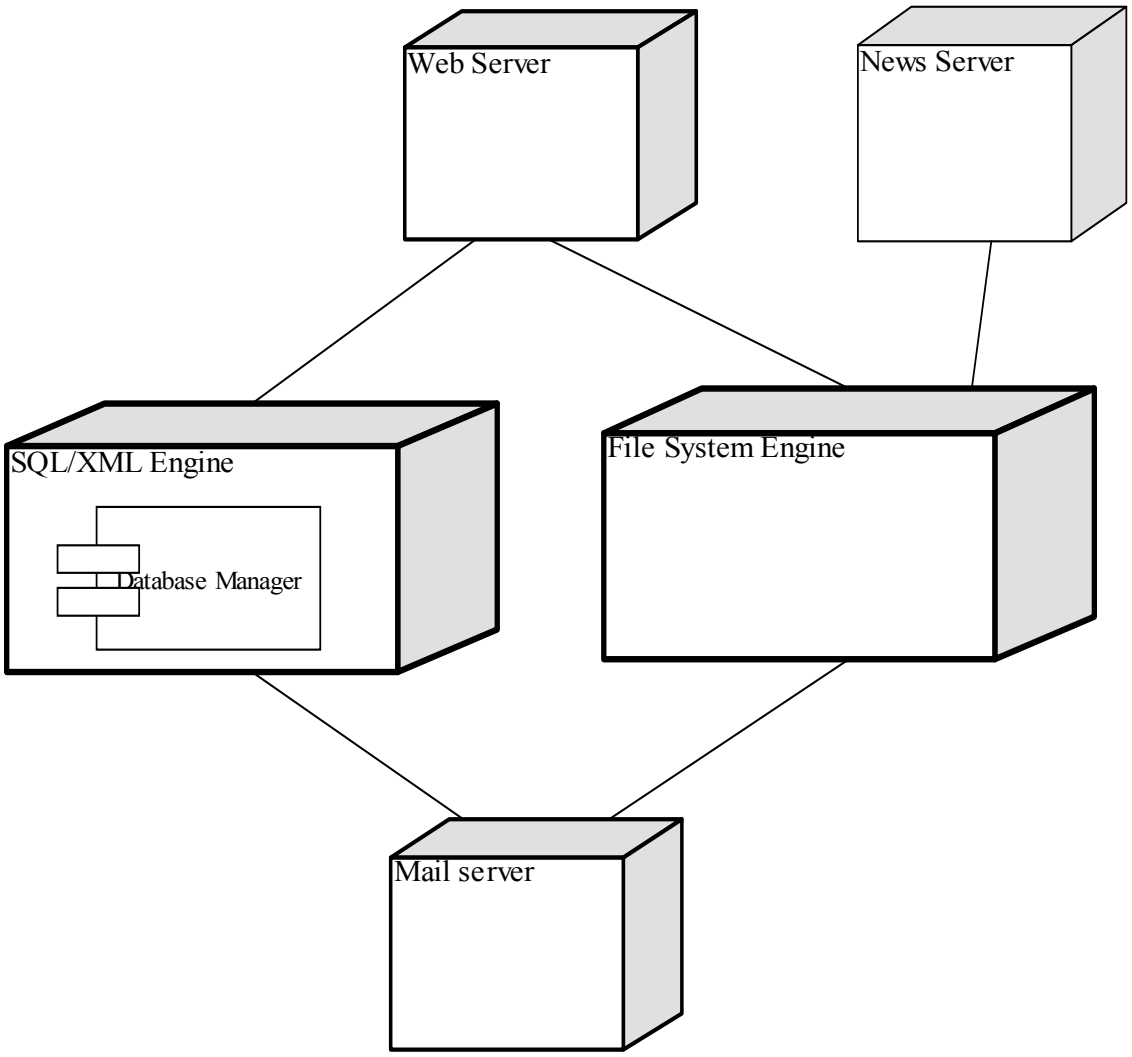
6.3.6 Activity Diagram of Searching



6.3.7 Activity Diagram for Mail Service



6.4 Deployment Diagram of the System



In deployment diagrams, servers are shown in a different nodes and associations among them is shown. Mail Server and Web Servers are both connected to SQL/XML Engine and File System Engine, whereas NNTP Server uses only File System Engine.

7. USER INTERFACE of the SYSTEM

Before constructing our user interface we examined several web pages that provides an interface for a NNTP servers and found many things that they have in common. That's why while designing our graphical user interface we decided to preserve the existing format of similar sites because most of the potential users are familiar with them.

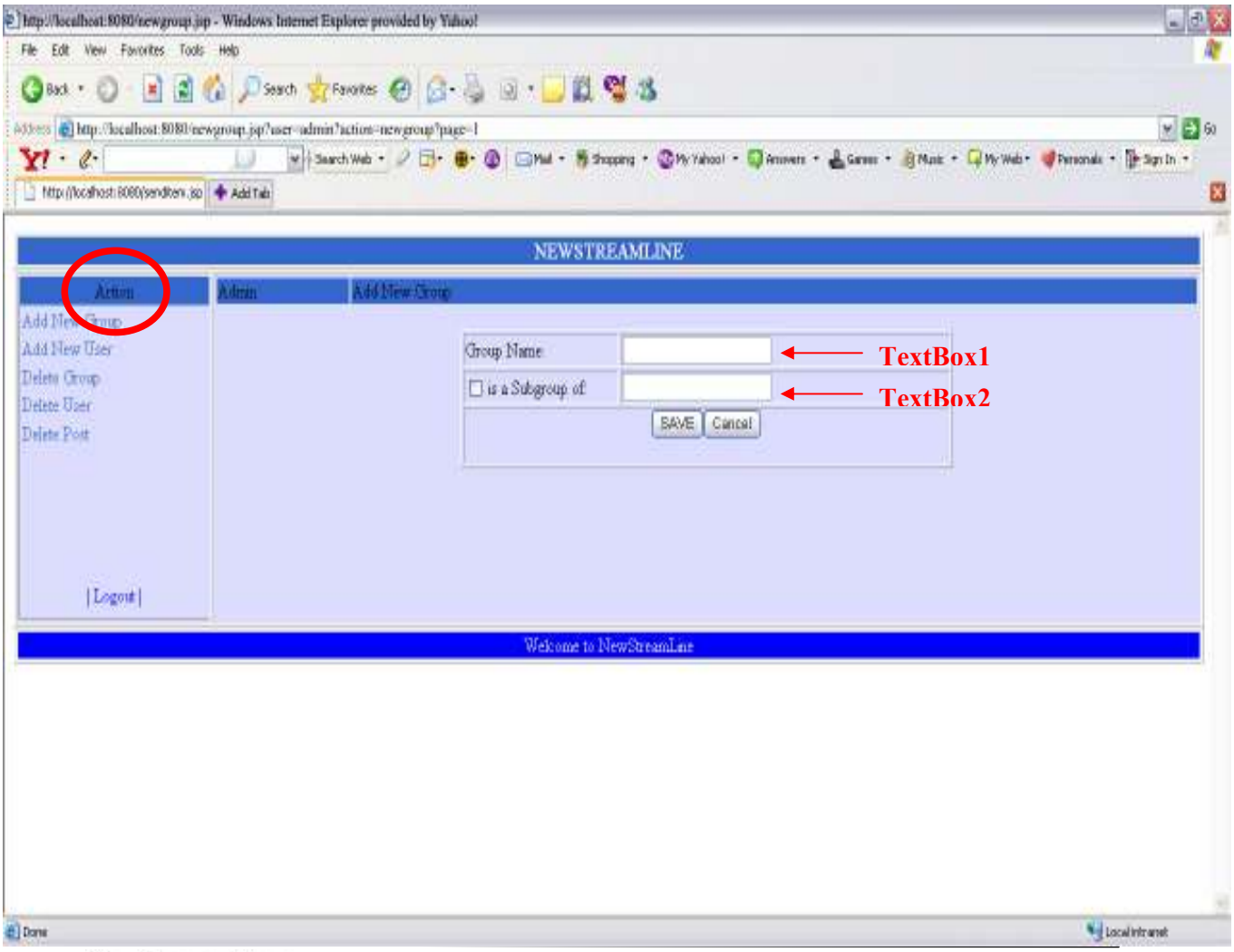
The goal is to keep GUI simple but in the same time very functional and coherent.

Our user interface is dived in two parts, one is for administrator, and the other is for users.

7.1 Administrator Graphical User Interface

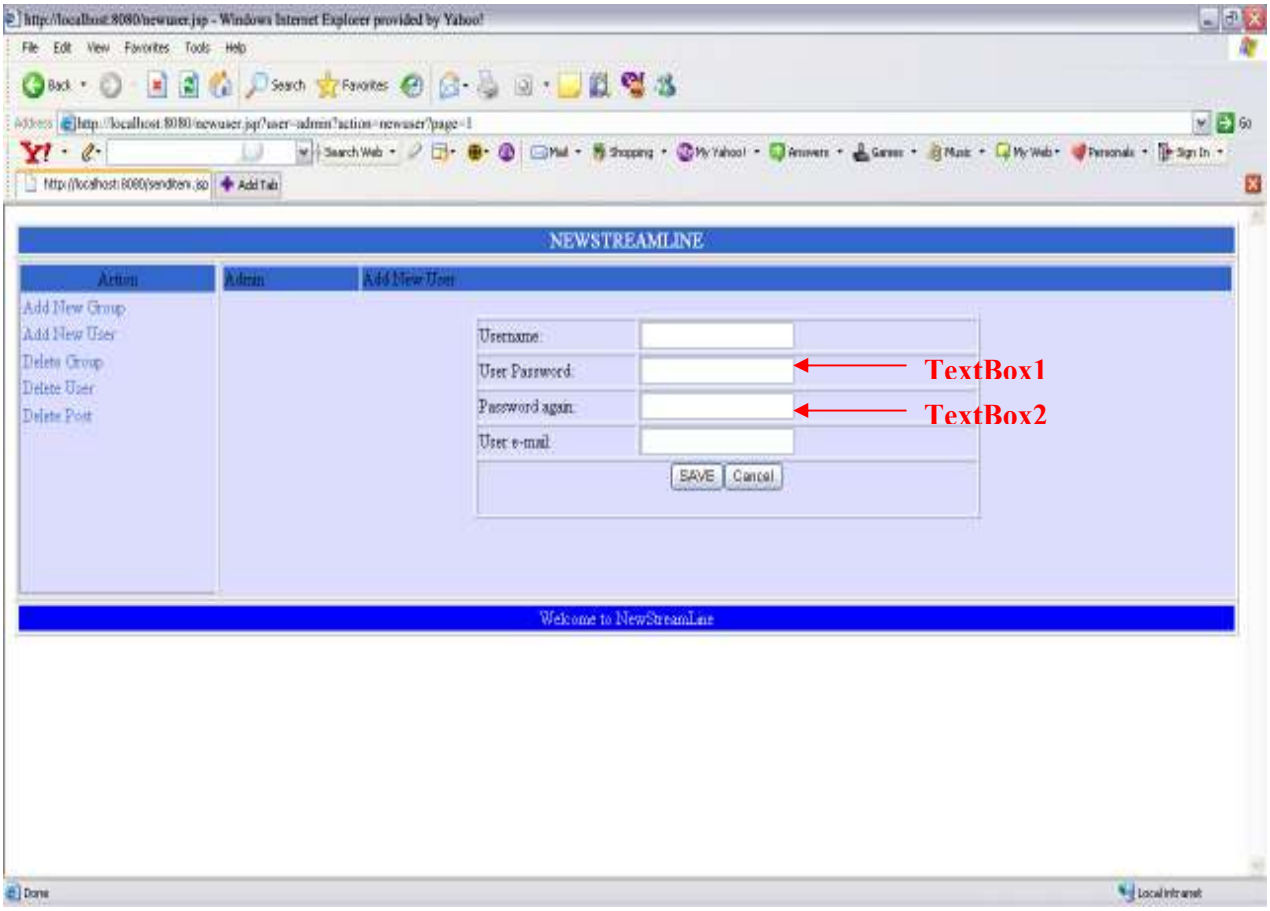
We omit showing some pages like logging in page or pages that are very similar to the shown one. However, in the sense of functionality, all pages have notification windows, error explanation labels and all that kind of thing that help users to understand system's work.

7.1.1 Adding New Group Page



In the TextBox1 Group name of new group is written. If you want to create a group that is a subgroup of another one, you must first check the checkbox near the TextBox2 and then write the name of the parent group in the TextBox2. After clicking the button SAVE system will check newly written groups for existence and act accordingly (for example will bring an error message if the group with the same group name already exists). By clicking on the button "Cancel" you will simple ignore the operation and return to the first page.

7.1.2 Adding New User



TextBox1 and TextBox2 are in the html type "password". When all information is entered the first thing done is matching these textboxes. If they do not match, error message is written. Then all fields are checked to be entered. When it's done, system checks all fields to be entered in appropriate format (for example e-mail address must contain "@" in it etc) After these, system connects to database which holds all user information, checks if the user with

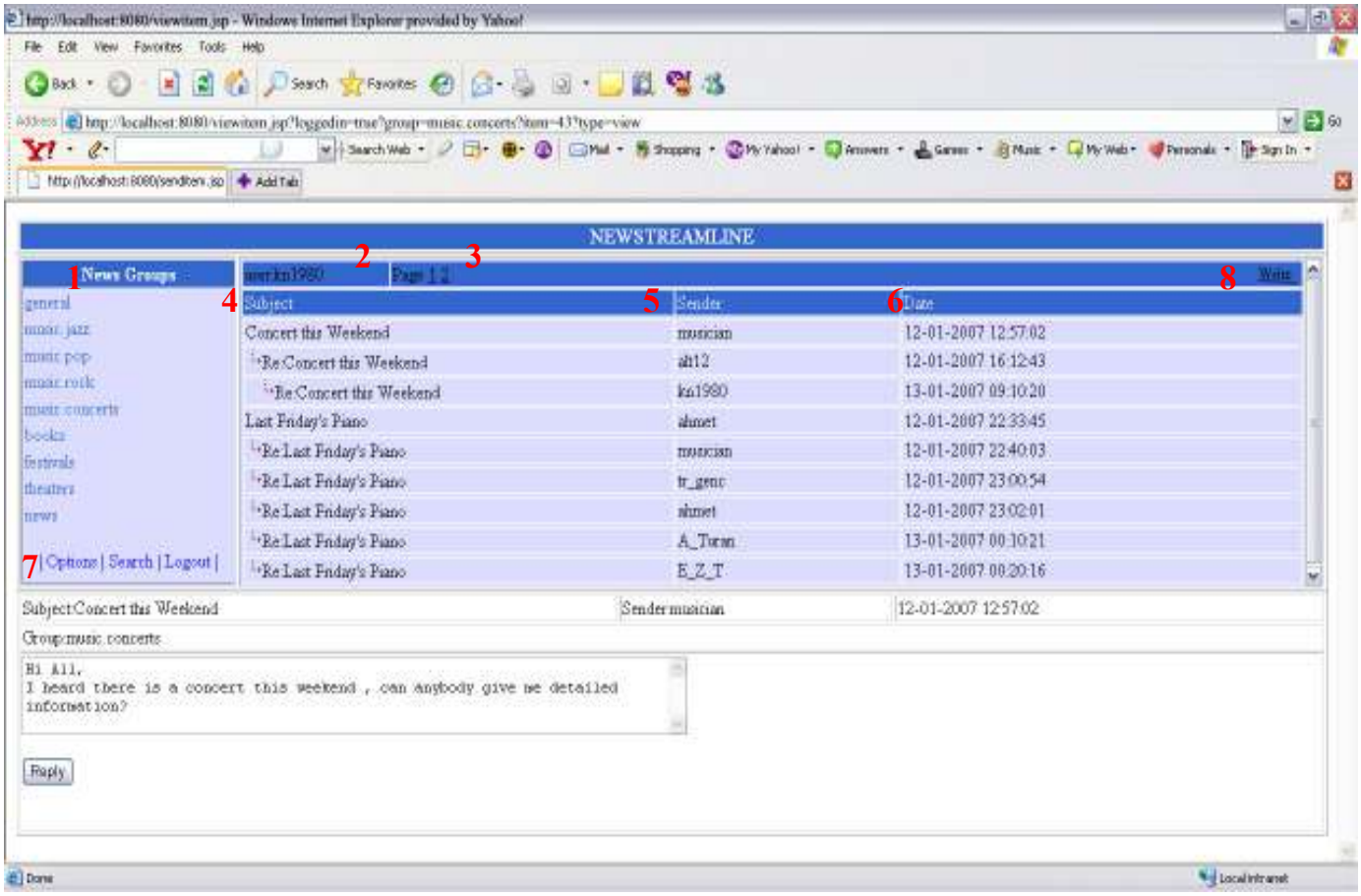
the same username exists. If yes, it writes notification and invites to enter new username. If not, it inserts new username and its information to database and notifies about successful creation of new user.

Last three actions have very simple GUI that's why they are not shown. To delete a newsgroup administrator must simple insert its name. It will be first checked for existence and than deleted. Similar procedure is followed when deleting user and post. They are first checked for existence and then deleted. Error message, notifications and all such kind information are printed all through the action.

7.2 User's Graphical User Interface.

Here we again omit showing simple pages, and display only pages which functionalities are more noticeable.

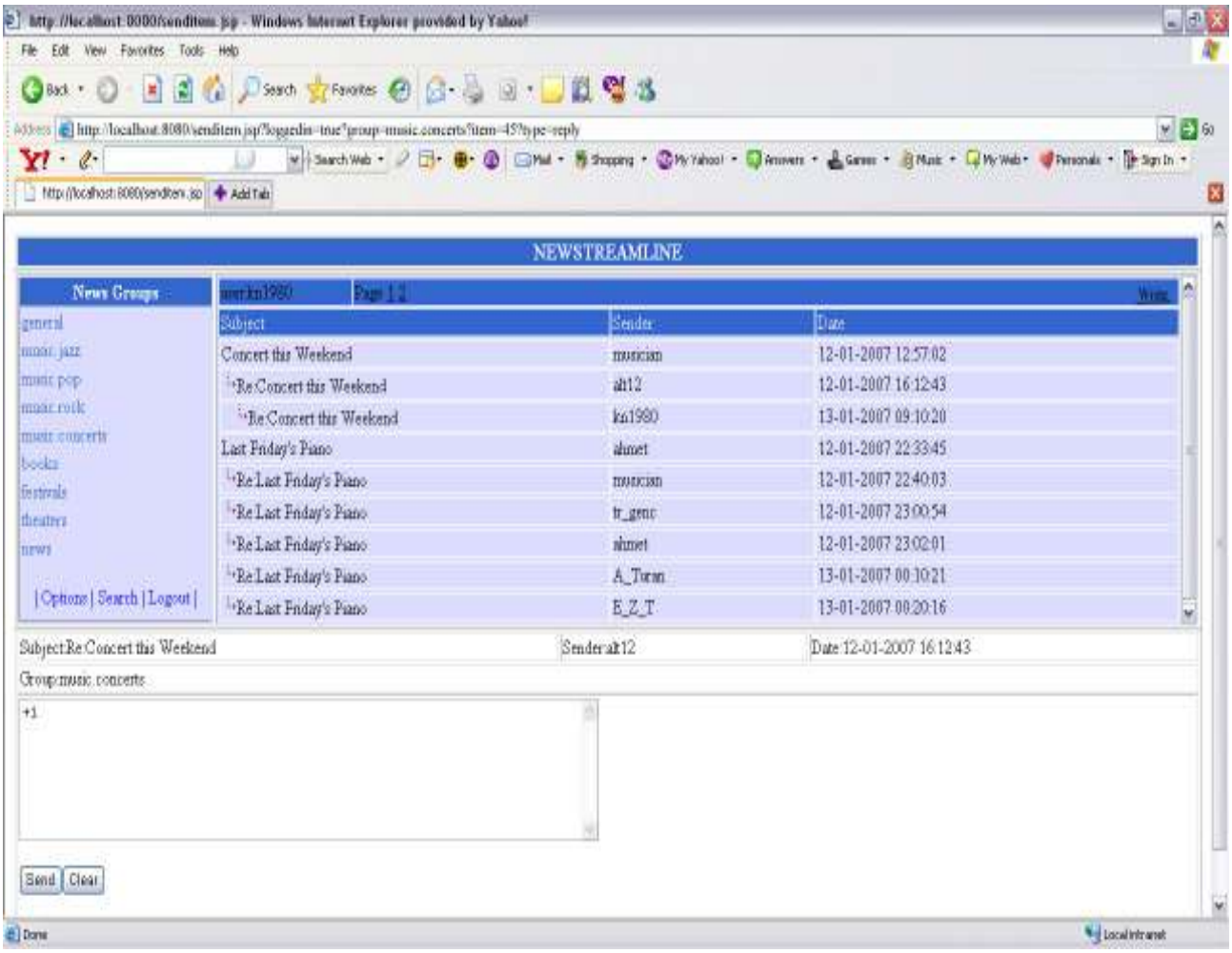
7.2.1. Viewing Post



News Group table (1) shows the news groups which are subscribed by user. It it's a div html space which has a scrolling vertical bar if needed. Field (2) shows the username of a user. Field (3) shows numbers of pages that exist in file system according to chosen group name. Column Subject (4) shows the post names. Reply posts are shown accordingly, deeper than the one which it is replied to. Column Sender (5) shows the name of the user, who sent the post, and column Date (6) shows the date and time when the post was posted. Field (7) shows s small menu items available for the user. By clicking Write link (8) User can write new post. User can view post simply by clicking on it. Post is shown in the bottom part of the page with

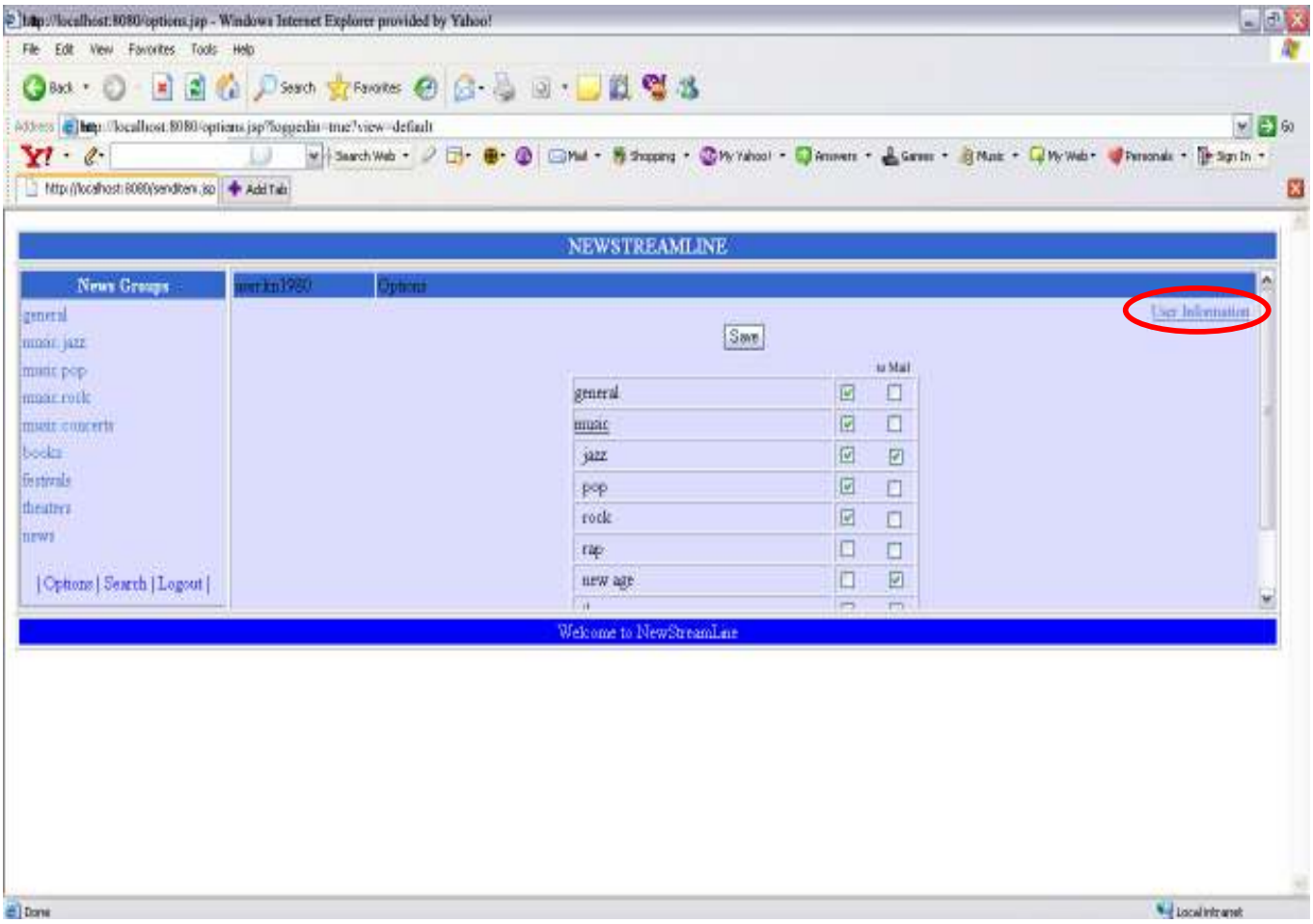
appropriate information about it. By clicking reply user can reply to the post that is being viewed.

7.2.2 Reply Post



After clicking Reply button page is redirected to new page that has a field for writing a post, that is a textbox within the form structure of html. In fields Subject, Sender and Date the information of post being replied is displayed. Subject Area can be changed and new title for a post can be written in that area. By clicking button "Send" post will be posted. All information bind to the post is send with it (such as sender, date etc) automatically. By clicking "Clear" text inside the textbox will be cleared. Write page is similar to reply that's why it is not shown.

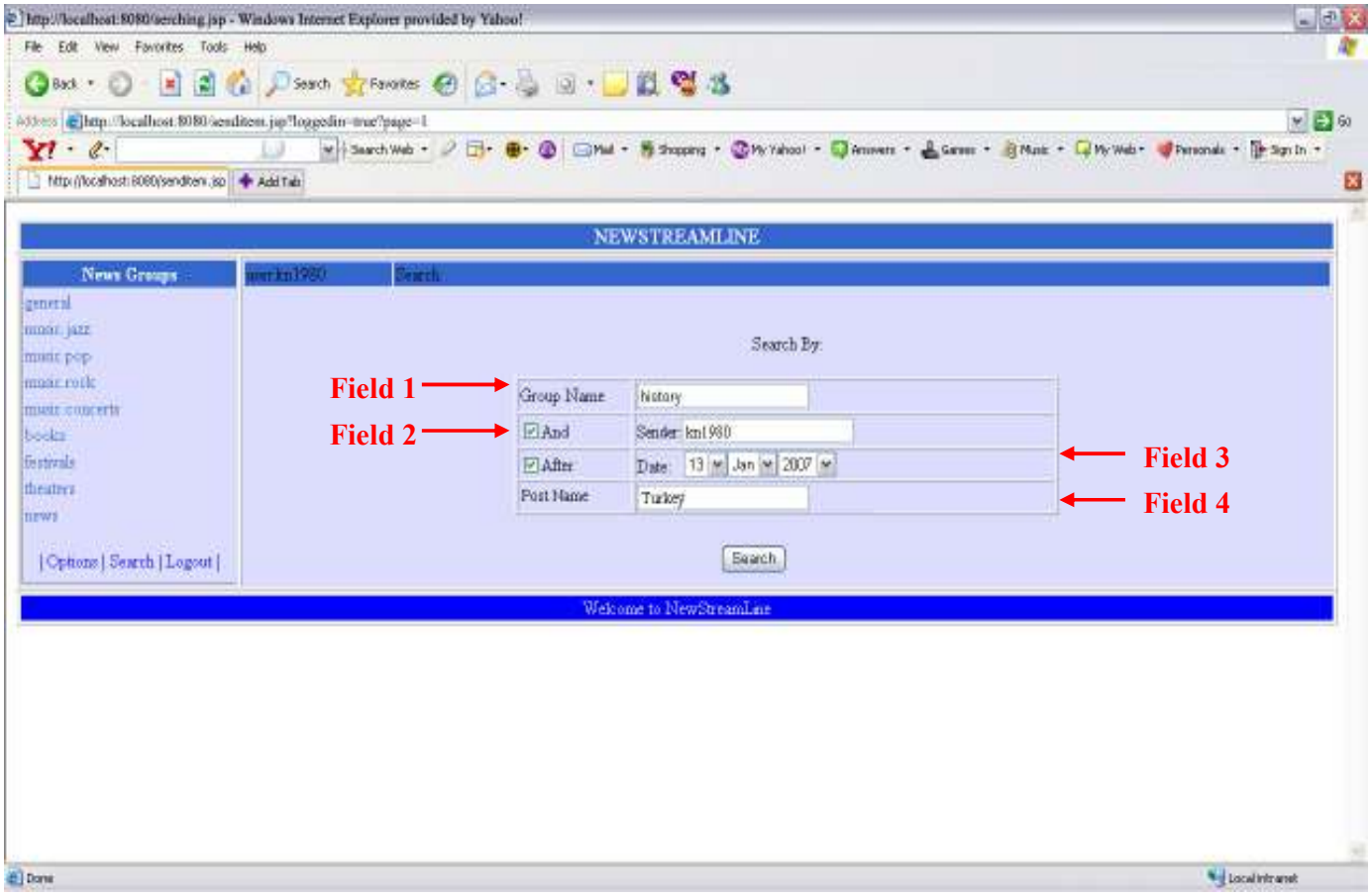
7.2.3 Options page



Options page provides user with ability of subscribing to a group and choosing a group that s/he wants to follow from his/her mail address. Checkboxes are used in order to record the specifications. After clicking “Save” button, information about preferences of user is saved in database of Mail_List and Group_List accordingly depending on a user id.

Also user can change User Information following the link on this page. We do not show it here because it is very similar to user page of administrator part. The difference is only that user can not change his username. But can change password and specify new e-mail address to receive mails of subscribed groups. All information about user is stored in database that’s why it’s updated if any changes are made.

7.2.4 Searching Page



In Searching page, by specifying some parameters user can search for a particular post. In Field 1 the name of the group for searching is entered. In Field 2, which is activated by checking the checkbox, the name of the sender can be specified. In Field 3 which is also activated by checking the checkbox one can specify the date, which the post was posted after. Date is specified using the drop down list to avoid the conflict between the day-month-year formats. In Field 4 the post name is entered. The return value of the engine is post view page containing the searched post, if found

8. SYNTAX SPECIFICATION

We have decided to obey to some standardization rules for the design and implementation of our project.

8.1 Syntax Conventions:

- Variables in global will have a name beginning with 'g_', so that they will be differentiated easily from the local ones. Moreover, variable names will have type conventions besides the global/local touchstones. To illustrate, "g_int_length" and "int_width" are valid variable names for global and local variables consecutively. Lastly, pointer variables will have a name beginning with 'p_'.
- Functions will have commented descriptive identification tags, clarifying the parameters and return types of functions.
- Class names will begin with an upper case letter; moreover, class names consisting of multiple words will have an upper case letter at the beginning of each word. For example, "User" and "NetworkAdmin" are valid class names.
- Attribute names and method names of classes will begin with a lower case letter; moreover, names consisting of multiple words will have an upper case letter at the beginning of each word.
- Database table names will begin with capital letters and the same procedure of class names will be valid for the names consisting of multiple words.

8.2 XML Representation

Here, we have provided the main skeleton of our XML files which enables the communication between servers and the connected clients. For further observation and details some explanatory XML can be found below:

```
<?xml version="1.0"?>
<!-- CONFIGURATION -->
<config>
  <James>
    <postmaster>newstreamline@googlegroups.com</postmaster>
    <servernames autodetect="true" autodetectIP="true">
      <servername>localhost</servername>
    </servernames>
    <usernames ignoreCase="true" enableAliases="true" enableForwarding="true"/>
    <inboxRepository>
      <repository destinationURL="file://var/mail/inboxes/" type="MAIL"/>
    </inboxRepository>
  </James>


 <spoolmanager>
    <threads> 10 </threads>

    <processor name="root">
      <mailet match="All" class="PostmasterAlias"/>
      <mailet match="RelayLimit=30" class="Null"/>
      </mailet>
      <mailet match="HasMailAttribute=spamChecked" class="ToProcessor">
        <processor> transport </processor>
      </mailet>
      <mailet match="All" class="SetMailAttribute">
        <spamChecked>true</spamChecked>
      </mailet>
```

```
      <mailet match="SMTPAuthSuccessful" class="ToProcessor">
        <processor> transport </processor>
      </mailet>
      <mailet match="InSpammerBlacklist=query.bondedsender.org."
            class="ToProcessor">
        <processor> transport </processor>
      </mailet>
      <mailet match="InSpammerBlacklist=dnsbl.njabl.org."
            class="ToProcessor">
        <processor> spam </processor>
      </mailet>
      <mailet match="InSpammerBlacklist=relays.ordb.org."
            class="ToProcessor">
        <processor> spam </processor>
      </mailet>
      <mailet match="All" class="ToProcessor">
        <processor> transport </processor>
      </mailet>
    </processor>

    <processor name="error">
      <mailet match="All" class="ToRepository">
        <repositoryPath> file://var/mail/error/</repositoryPath>
      </mailet>
    </processor>

    <processor name="transport">
      <mailet match="SMTPAuthSuccessful" class="SetMimeHeader">
        <name>X-UserIsAuth</name>
        <value>true</value>
      </mailet>
      <mailet match="HasMailAttribute=org.apache.james.SMIMECheckSignature"
class="SetMimeHeader">
        <name>X-WasSigned</name>
```

```
        <value>true</value>
      </mailet>
      <mailet match="RecipientIsLocal" class="LocalDelivery"/>
      <mailet match="HostIsLocal" class="ToProcessor">
        <processor> local-address-error </processor>
      </mailet>
      <mailet match="RemoteAddrNotInNetwork=127.0.0.1" class="ToProcessor">
        <processor> relay-denied </processor>
        <notice>550 - Requested action not taken: relaying denied</notice>
      </mailet>
      <mailet match="All" class="RemoteDelivery">
        <outgoing> file://var/mail/outgoing/ </outgoing>
        <delayTime>  5 minutes </delayTime>
        <delayTime> 10 minutes </delayTime>
        <delayTime> 45 minutes </delayTime>
        <delayTime>  2 hours </delayTime>
        <delayTime>  3 hours </delayTime>
        <delayTime>  6 hours </delayTime>
        <maxRetries> 25 </maxRetries>
        <deliveryThreads> 1 </deliveryThreads>
        <sendpartial>false</sendpartial>
        <bounceProcessor>bounces</bounceProcessor>
      </mailet>
    </processor>
  </spoolmanager>

  <dnsserver>
    <servers>
       <server>127.0.0.1</server>
    </servers>
    <autodiscover>true</autodiscover>
    <authoritative>false</authoritative>
    <maxcachesize>50000</maxcachesize>
  </dnsserver>
```

```
<remotemanager enabled="true">
  <port>4555</port>
  <handler>
    <helloName autodetect="true">myMailServer</helloName>
    <administrator_accounts>
      <account login="root" password="root"/>
    </administrator_accounts>
    <connectiontimeout> 60000 </connectiontimeout>
  </handler>
</remotemanager>

<smtpserver enabled="true">
  <port>25</port>
  <useTLS>true</useTLS>
  <handler>
    <helloName autodetect="true">myMailServer</helloName>
    <connectiontimeout>360000</connectiontimeout>
    <authRequired>true</authRequired>
    <authorizedAddresses>127.0.0.0/8</authorizedAddresses>
    <verifyIdentity>true</verifyIdentity>
    <maxmessagesize>0</maxmessagesize>
  </handler>
</smtpserver>
<nntpserver enabled="true">
  <port>119</port>
  <handler>
    <helloName autodetect="true">myMailServer</helloName>
    <connectiontimeout>120000</connectiontimeout>
    <authRequired>false</authRequired>
  </handler>
</nntpserver>
```

```
  <nntp-repository>
    <readOnly>false</readOnly>

    <rootPath>file://var/nntp/groups</rootPath>
    <tempPath>file://var/nntp/temp</tempPath>
    <articleIDPath>file://var/nntp/articleid</articleIDPath>
    <articleIDDomainSuffix>news.james.apache.org</articleIDDomainSuffix>

<!--NEWSGROUPS CAN BE SPECIFIED HERE-->
    <newsgroups>
      <newsgroup>NewStreamLine</newsgroup>
    </newsgroups>

    <spool>
      <configuration>
        <spoolPath>file://var/nntp/spool</spoolPath>
        <threadCount>1</threadCount>
        <threadIdleTime>60000</threadIdleTime>
      </configuration>
    </spool>
  </nntp-repository>
  <spoolrepository destinationURL="file://var/mail/spool/" type="SPOOL"/>

  <mailstore>
    <repositories>
      <repository class="org.apache.james.mailrepository.AvalonMailRepository">
        <protocols>
          <protocol>file</protocol>
        </protocols>
        <types>
          <type>MAIL</type>
        </types>
      </repository>
      <repository class="org.apache.james.mailrepository.AvalonSpoolRepository">
```

```
    <protocols>
      <protocol>file</protocol>
    </protocols>
  </repository>
  <repository class="org.apache.james.mailrepository.JDBCMailRepository">
    <protocols>
      <protocol>db</protocol>
    </protocols>
    <types>
      <type>MAIL</type>
    </types>
    <config>
      <sqlFile>file://conf/sqlResources.xml</sqlFile>
    </config>
  </repository>
  <repository class="org.apache.james.mailrepository.JDBCSpoolRepository">
    <protocols>
      <protocol>db</protocol>
    </protocols>

  <!--SQL MAIL REPOSITORY CAN BE SPECIFIED HERE-->
    <config>
      <sqlFile>file://conf/sqlResources.xml</sqlFile>
      <maxcache>1000</maxcache>
    </config>
  </repository>
  <repository class="org.apache.james.mailrepository.JDBCMailRepository">
    <protocols>
      <protocol>dbfile</protocol>
    </protocols>
    <types>
      <type>MAIL</type>
    </types>
    <config>
```

```
            <sqlFile>file://conf/sqlResources.xml</sqlFile>
            <filestore>file://var/dbmail</filestore>
          </config>
        </repository>
        <repository class="org.apache.james.mailrepository.JDBCSpoolRepository">
          <protocols>
            <protocol>dbfile</protocol>
          </protocols>
          <config>
            <sqlFile>file://conf/sqlResources.xml</sqlFile>
            <filestore>file://var/dbmail</filestore>
            <maxcache>1000</maxcache>
          </config>
        </repository>
        <repository class="org.apache.james.mailrepository.MBoxMailRepository">
          <protocols>
            <protocol>mbox</protocol>
          </protocols>
          <types>
            <type>MAIL</type>
          </types>
        </repository>
        <repository
class="org.apache.james.mailrepository.filepair.File_Persistent_Object_Repository">
          <protocols>
            <protocol>file</protocol>
          </protocols>
          <types>
            <type>OBJECT</type>
          </types>
          <models>
            <model>SYNCHRONOUS</model>
            <model>ASYNCHRONOUS</model>
            <model>CACHE</model>
```

```
        </models>
      </repository>
      <repository
class="org.apache.james.mailrepository.filepair.File_Persistent_Stream_Repository">
        <protocols>
          <protocol>file</protocol>
        </protocols>
        <types>
          <type>STREAM</type>
        </types>
        <models>
          <model>SYNCHRONOUS</model>
          <model>ASYNCHRONOUS</model>
          <model>CACHE</model>
        </models>
      </repository>
    </repositories>
  </mailstore>
  <users-store>
    <repository name="LocalUsers"
class="org.apache.james.userrepository.UsersFileRepository">
      <destination URL="file://var/users/"/>
    </repository>
    <repository name="LocalUsers"
class="org.apache.james.userrepository.JamesUsersJdbcRepository"
destinationURL="db://maildb/users">
      <sqlFile>file://conf/sqlResources.xml</sqlFile>
    </repository>
     &listserverStores;
  </users-store>

  <database-connections>
    <data-source name="maildb" class="org.apache.james.util.dbcp.JdbcDataSource">
      <driver>com.inet.tds.TdsDriver</driver>
```

```
        <dburl>jdbc:inetdae7:127.0.0.1?database=NSL</dburl>
        <user>NSL_user</user>
        <password>newstreamline</password>
        <max>20</max>
      </data-source>
  </database-connections>

  <connections>
    <idle-timeout>300000</idle-timeout>
    <max-connections>30</max-connections>
  </connections>

  <sockets>
    <server-sockets>
      <factory name="plain"
class="org.apache.avalon.cornerstone.blocks.sockets.DefaultServerSocketFactory"/>
    </server-sockets>
    <client-sockets>
      <factory name="plain"
class="org.apache.avalon.cornerstone.blocks.sockets.DefaultSocketFactory"/>
    </client-sockets>
  </sockets>

  <thread-manager>
    <thread-group>
      <name>default</name>
      <priority>5</priority>
      <is-daemon>false</is-daemon>
      <max-threads>100</max-threads>
      <min-threads>20</min-threads>
      <min-spare-threads>20</min-spare-threads>
    </thread-group>
  </thread-manager>
</config>
```

```
<!--  SERVER -->
<server>
   <factories>
     <factory type="file"
class="org.apache.avalon.excalibur.logger.factory.FileTargetFactory"/>
   </factories>

   <categories>
     <category name="" log-level="INFO">
       <log-target id-ref="default"/>
     </category>
     <category name="James.Mailet" log-level="INFO">
       <log-target id-ref="James-Mailet-target"/>
     </category>
     <category name="James" log-level="INFO">
       <log-target id-ref="James-target"/>
     </category>
     <category name="spoolmanager" log-level="INFO">
       <log-target id-ref="spoolmanager-target"/>
     </category>
     <category name="dnsserver" log-level="INFO">
       <log-target id-ref="dnsserver-target"/>
     </category>
     <category name="remotemanager" log-level="INFO">
       <log-target id-ref="remotemanager-target"/>
     </category>
     <category name="pop3server" log-level="INFO">
       <log-target id-ref="pop3server-target"/>
     </category>
     <category name="smtpserver" log-level="INFO">
       <log-target id-ref="smtpserver-target"/>
     </category>
     <category name="nntpserver" log-level="INFO">
```

```
      <log-target id-ref="nntpserver-target"/>
    </category>
    <category name="nntp-repository" log-level="INFO">
      <log-target id-ref="nntp-repository-target"/>
    </category>
    <category name="mailstore" log-level="INFO">
      <log-target id-ref="mailstore-target"/>
    </category>
    <category name="users-store" log-level="INFO">
      <log-target id-ref="users-store-target"/>
    </category>
    <category name="objectstorage" log-level="INFO">
      <log-target id-ref="objectstorage-target"/>
    </category>
    <category name="connections" log-level="INFO">
      <log-target id-ref="connections-target"/>
    </category>
    <category name="sockets" log-level="INFO">
      <log-target id-ref="sockets-target"/>
    </category>
    <category name="scheduler" log-level="INFO">
      <log-target id-ref="scheduler-target"/>
    </category>
    <category name="fetchmail" log-level="INFO">
      <log-target id-ref="fetchmail-target"/>
    </category>
   </categories>
   </targets>
  </logs>
</server>
</targets>
</logs>
</server>
```

9. PDL MODEL

```
Main()
SET DEFAULTS
LOOP WHILE ListenMode
IF (int_request) THEN
        IF(not int_queue)
                DO ServeUserRequest(int_userRequest)
        ELSE push the request to the existing queue
                serve to first pending request from queue
        ENDIF
ENDIF
END LOOP
ServeUserRequest(int_request)
IF (int_userRequest) THEN
        check the rules to be applied to that user
        IF(int_userBlocked) THEN
                int_action < - - NONE
        ELSE int_Action < - - SERVE
        ENDIF
        Serve(int_request)
ENDIF
Serve(int_request)
LOOP WHILE ( not int_request NONE)
        DO invoke Web Service
        parse the int_request
        IF (int_request is RSS related) THEN
                invoke XML Service
                update the request
        ELSE IF (int_request is Email related) THEN
                invoke SMTP Service
                update the request
        ELSE IF (int_request is News related) THEN
                invoke NNTP Service
                update the request
        ENDIF
```

ENDIF
parse the request
ENDLOOP

10. CONCLUSION

Throughout this document we described in details the work of the NewStreamLine news exchange server. All differences done to the system after initial design report submission were shown and explained accordingly. Different types of diagrams were used in order to show the workflow of the system. Firstly, we have added System Architecture Part that explains modules in general aspect. Moreover we added an architecture diagram of the system to explain different modules graphically. Secondly, Use Case diagrams were dropped of because of Object- Oriented Software Design rules which use UML for modeling. In addition to these, we have updated all UML diagrams according to the changes that were done to the system. Thirdly, Graphical User Interface was constructed for the Web Page of the NewStreamLine and most important pages were shown in this document. Fourthly, as an XML configuration files are very important in any kind of server including software, we decided to include the part of config.xml file of our server in the design report. Finally, PDA model that includes general functionality of the server was written.

We are sure that this report clarified all points which were not clear in all previous reports.

11. APPENDIX - GANNT CHART