EZ Intranet Messenger

GitHub Link: https://github.com/EyeGore/SOEN343

Tamyres Sayegh Ezarani Guimaraes	*7187858*
Vicranth Somasundaram	*6609759*
Goaba Mogapi	*6018211*
Dan Zhang	*6733298*
Sarah Schneider	*7619049*
Igor Firman	*6033539*

Summary of Project

EZIM is an instant messenger application for the intranet network that supports online status notification, file transmission and real time chat with computers connected via LAN. It does not need a centralized server, instead it just finds all other users within the same network by obtaining their IP address from the DHCP server. It runs on a Java Virtual Machine, therefore it works on Windows, Mac OS X and Linux platforms.

Class Diagram of Actual System

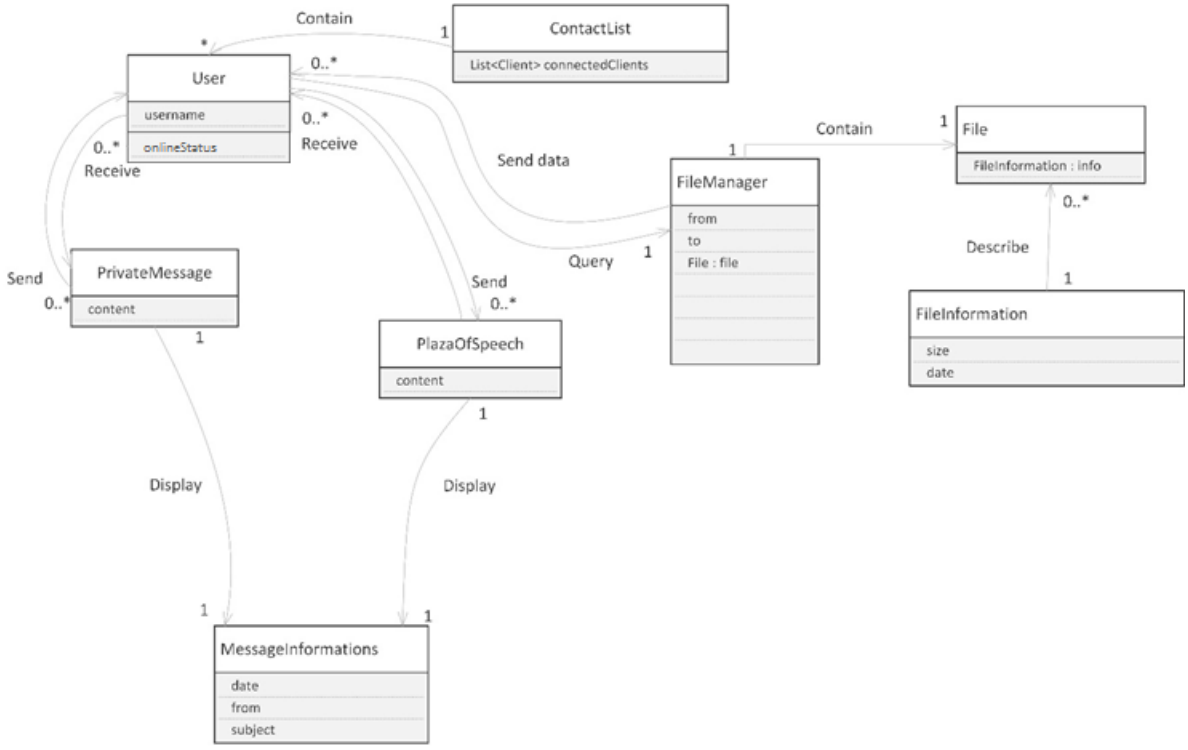


Figure 1: Conceptual Diagram

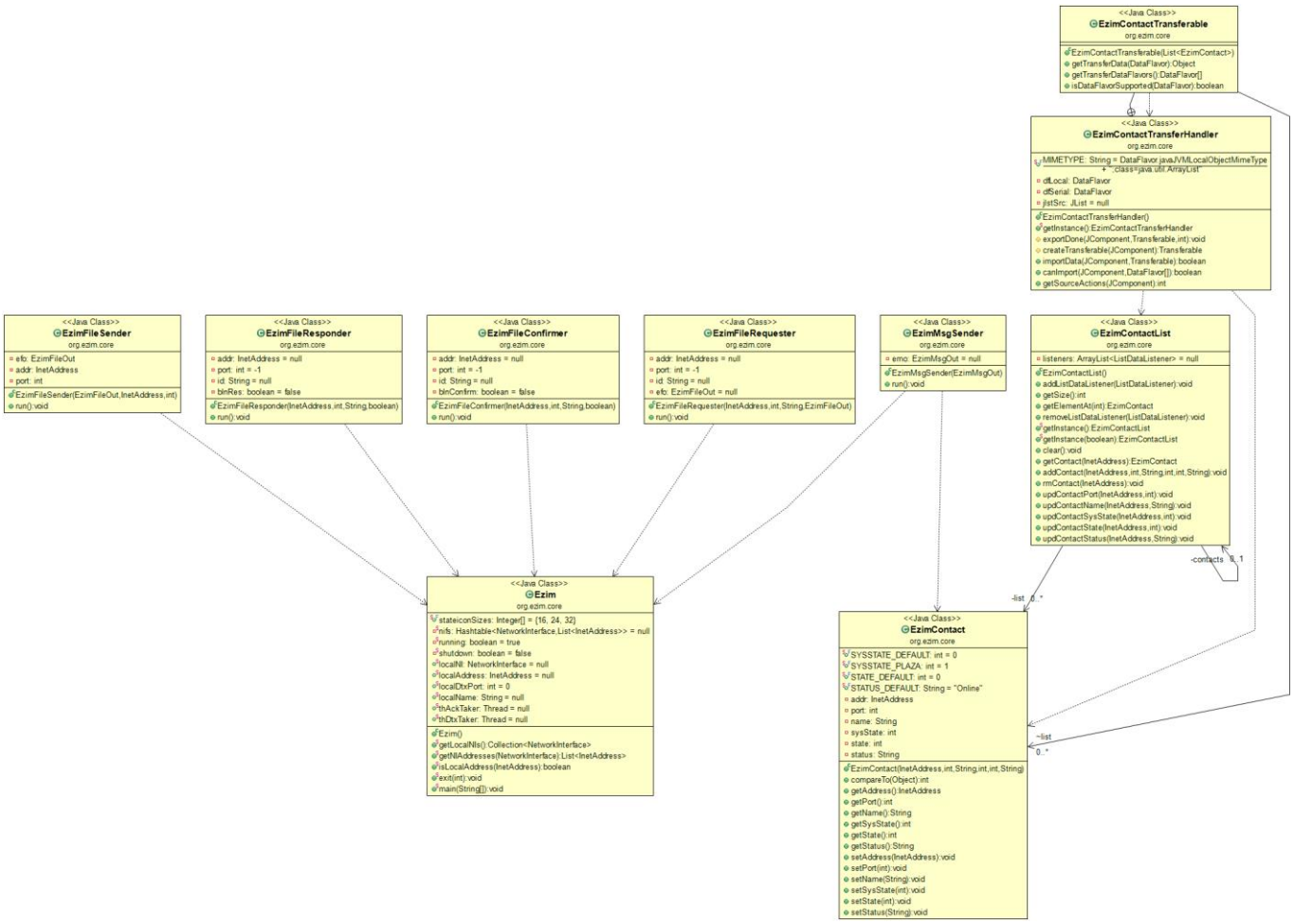


Figure 2: UML diagram of classes of interest

Description of the Classes

Overview: User enters into EZIM application [EZIM] application and sees a contact list [EzimContactList] which displays other online users of the EZIM application in the local network. From the contact list, a user can select other users to send messages [EzimMsgSender] or files [EzimFileSender].

Ezim and EzimContact

Ezim is the start of application in which the main method is contained. This class has relationships with most of the other classes in its system. The EzimContact is mainly used for creating a new instant of a contact and statuses.

EzimContactList

This is class is responsible for listing all the contacts collaborating with the Ezim class. The contact list can show list of users you can communicate with.

EzimMsgSender

The EzimMsgSender class' purpose is to create and send emails to a contact from the contact. An EzimMsgSender can send email at most 1 EzimContact at a time. An Ezim is composed of one EzimMsgSender.

EzimFileSender

The EzimFileSender class' purpose is to send files to the contact by figuring out the address and ports of recipients. It also verifies if the files are accepted by the **EzimFileResponder**. Once the file transmission is complete, the **EzimFileConfirmer** will confirm the completion. EzimFileSender can send any amount of files to EzimContact though the Ezim class.

- The **EzimFileResponder** class' purpose is to indicate whether the file transmission was accepted by the recipient or not.
- The **EzimFileConfirmer** class' purpose is to provide the methods to confirm that the file transmission is done or not.

Conceptual Class	**Actual Class**
User	EzimContact,Ezim
ContactList	EzimContactLiST
PrivateMessage	EzimMsgSender
FileManager	EzilmFileSender, EzimFileResponder,EzimFileConfirmer

Description of ObjectAid UML Explorer

ObjectAid UML Explorer for eclipse was used to generate the class diagram from the source code. It is easy to install and fairly easy to use. Once ObjectAid UML Explorer plug-in is installed, you choose to create a new class diagram from new project menu. You select the options for

the class diagram, such as what relationships to show and overall detail of the diagram. Once the new class diagram is created, you drag and drop the classes from the source code that are automatically translated into a UML diagram. There is one disadvantage, if you want to refine the class diagram to show only pertinent attributes/methods, there is no way to remove it from the generate diagram without actually commenting out attribute/method in question inside the source code.

Class Relationships

```
package org.ezim.core;

public class EzimContactTransferHandler
        extends TransferHandler
{       protected Transferable createTransferable(JComponent jcIn)
        {
                EzimContactTransferable ectOut = null;
                if
                (jcIn instanceof JList
                        && ((JList<?>) jcIn).getModel() instanceof EzimContactList
                )
                {
                        @SuppressWarnings("unchecked")
                        JList<EzimContact> jlstEc = (JList<EzimContact>) jcIn;

                        ectOut = new EzimContactTransferable
                        (
                                jlstEc.getSelectedValuesList()
                        );
                }
                return ectOut;
        }
```

//INTERNAL CLASS----------

```
        public class EzimContactTransferable
                implements Transferable
```

```
	{
		ArrayList<EzimContact> list;
		public EzimContactTransferable(List<EzimContact> lIn)
		{.list = new ArrayList<EzimContact>(lIn);
		}
	}
}
```

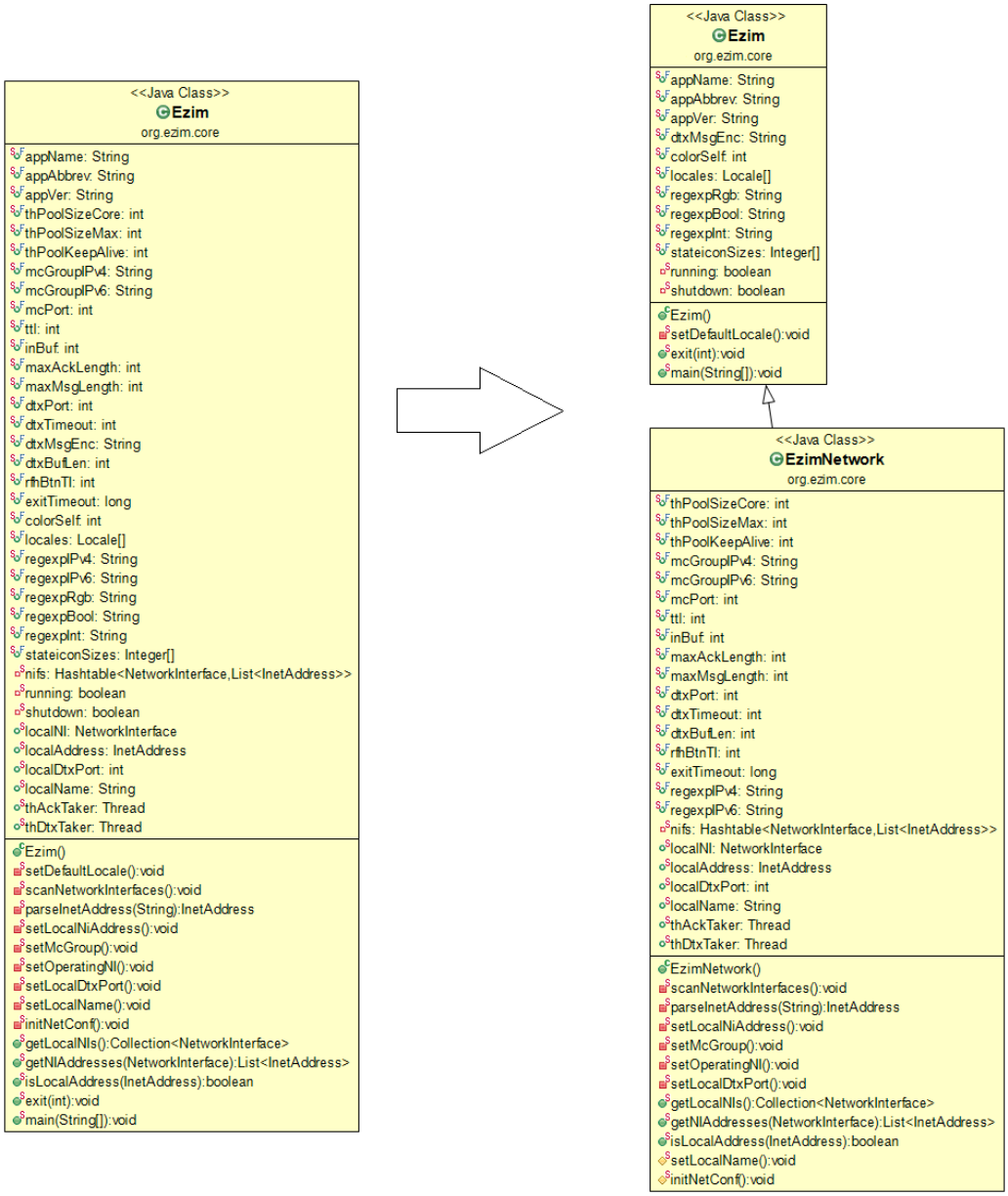
Code Smells an System Level Refactorings

God Class

Ezim is the starting point of the application. It contains application constants such as name and version, but it also manages network settings, reducing cohesion and making it less readable and maintainable. If changes have to be made to the network information and to the related methods, it's not very intuitive to look for those methods in the **Ezim** class.

To solve this issue, we can refactor by adding a new class: **EzimNetwork**, which will contain the network data members such as *tdPoolSizeMax* and methods such as scanNetworkInterfaces(), and keep **Ezim** for the application metadata such as *appName* and the entry and exit points main() and exit().

All references throughout the program to the data members and methods that previously belonged to the **Ezim** class have to be renamed and now reference the new class **EzimNetwork** instead.



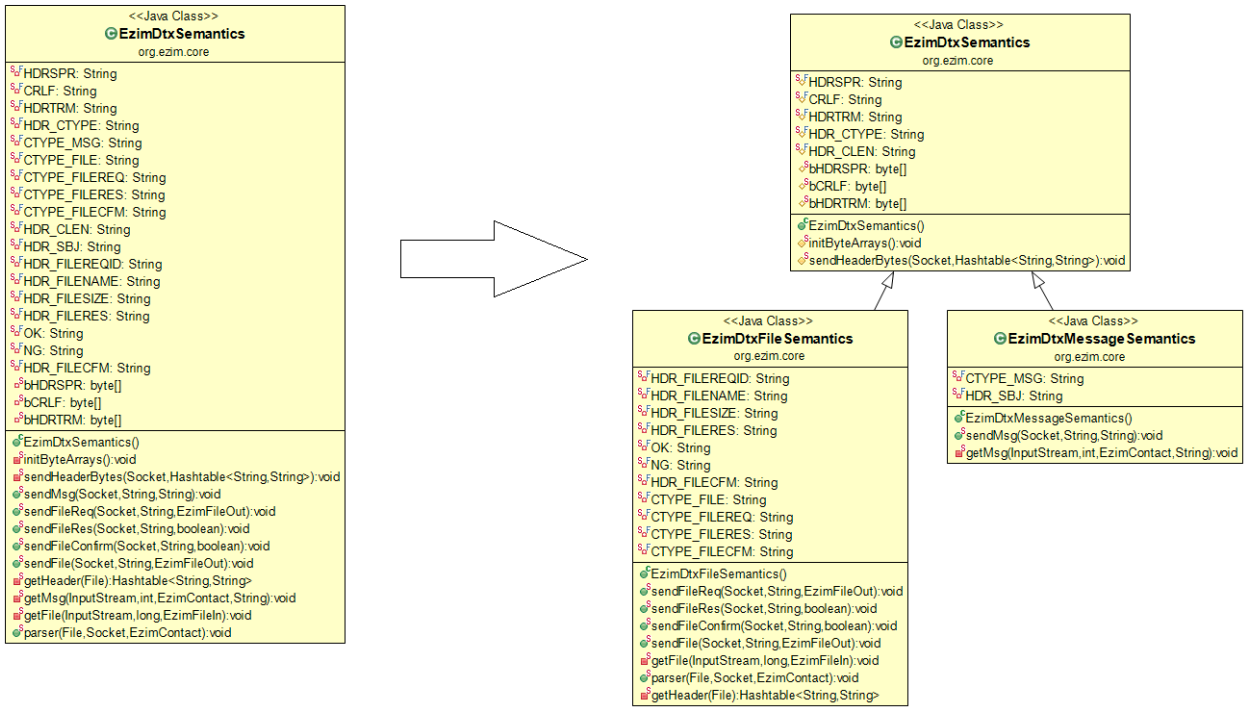
Large Class

EzimDtxSemantics is a large class (892 lines of code) and is responsible for sending and retrieving messages and files, therefore it has two distinct responsibilities.

It can be split into two subclasses for increased cohesion: one for message treatment and another for file treatment. The first new subclass will be called **EzimDtxFileSemantics** and will contain data members related to files such as *HDR_FILESIZE* and methods such as getFile(). The second subclass will be called **EzimDtxMessageSemantics** and will contain message related data members such as *CTYPE_MSG* and methods such as getMsg().

Everything that is not related specifically to messages or files and can be applied to both, can stay in the parent class. Doing so will increase cohesion and make the code more maintainable.

Like on the previous code smell, references to data members and methods that belonged previously to **EzimDtxSemantics** throughout the program have to be renamed and reference their new locations.



Duplicated code

On **EzimContactList**, the same lines of code (a nested if statement) appear in several methods. They are meant for testing something specific, and if for some reason this test needs to be changed, the person responsible for maintaining the code has to look for every occurrence of it and then make the necessary changes. It can take time and is prone to many problems such as inconsistent code.

This can be solved by extracting the lines that are being repeated (highlighted on the screenshot below) and placing them in a method of their own, which will return a Boolean . This refactoring will result in only one if statement (being the newly created method the condition to be evaluated) and less duplicated code. The condition being evaluated is also going to become more clear because the name of the method will be self-explanatory, unlike the way it currently is.

```
public EzimContact getContact(InetAddress iaIn)
{
    EzimContact ecOut = null;
    InetAddress iaTmp = this.normalizeAddressIfLocal(iaIn);
    int iIdx = idxContact(iaTmp);

    if (iIdx > -1)
    {
        ecOut = this.getElementAt(iIdx);
        if(ecOut != null && iaTmp.equals(iaIn) && ! ecOut.getAddress().equals(iaTmp))
        {
            ecOut = null;
        }
    }

    return ecOut;
}
```

Specific Refactorings

The specific refactorings that we are going to implement in Milestone 4 include the following:

1. Doing refactoring on a Large Class called EzimDtxSemantics which is over 800 lines of code. We are going to extract some smaller classes which are dealing with files (including methods of sendFileReq(), sendFileRes(), sendFileConfirm(), sendFile(), and getFile()), and dealing with messages (including methods of sendMSG() and getMSG()) so that the class will be shorter and more cohesive.
2. Doing refactoring on duplicated code (pasted in the next page) in four methods called updContactName(InetAddress, String), updContactSysState(InetAddress, int), updContactState(InetAddress, int), and updContactStatus(InetAddress, strStatus) in EzimContactList Class. We will use extract methods to implement the refactoring.
3. Fixing complex if statements in method setLocalNiAddress() in Ezim class. setLocalNiAddress() is a very long method (169 lines of code), so we can also claim that we will be doing refactoring for a long method.

```
public void updContactName(InetAddress iaIn, String strName)
        {
                InetAddress iaTmp = this.normalizeAddressIfLocal(iaIn);
                int iIdx = this.idxContact(iaTmp);

                if (iIdx > -1)
                {EzimContact ecTmp = this.getElementAt(iIdx);
            if (ecTmp != null && ecTmp.getAddress().equals(iaTmp))
                        {……}
                }
        }



        public void updContactSysState(InetAddress iaIn, int iState)
        {    InetAddress iaTmp = this.normalizeAddressIfLocal(iaIn);
                int iIdx = this.idxContact(iaTmp);

                if (iIdx > -1)
                {EzimContact ecTmp = this.getElementAt(iIdx);
             if (ecTmp != null && ecTmp.getAddress().equals(iaTmp))
                        {……}
                }
        }



    public void updContactState(InetAddress iaIn, int iState)
        {
                InetAddress iaTmp = this.normalizeAddressIfLocal(iaIn);
                int iIdx = this.idxContact(iaTmp);

                if (iIdx > -1)
                {   EzimContact ecTmp = this.getElementAt(iIdx);
            if (ecTmp != null && ecTmp.getAddress().equals(iaTmp))
                        {……}
                }
        }



public void updContactStatus(InetAddress iaIn, String strStatus)
        {    InetAddress iaTmp = this.normalizeAddressIfLocal(iaIn);
                int iIdx = this.idxContact(iaTmp);

                if (iIdx > -1)
                {EzimContact ecTmp = this.getElementAt(iIdx);
             if (ecTmp != null && ecTmp.getAddress().equals(iaTmp))
                        {
                                ……
                        }
                }
        }
```