Facebook chat Instant Messenger (FBCIM)

Bala Arun Reddy Vatti 6228569 balaarunreddyv@gmail.com

Deepu Kumar 5972515 Kumar.deep@live.in

Pankaj Kapania 6249922 pankajkapania@yahoo.com

Paramjeet Singh 1370189 er.paramjeetsingh@yahoo.com

Gilles Desrochers 1255185 gilles.desrochers@hotmail.com

**SOEN 6471 – Advanced software Architecture**

Professor Peter Rigby

Milestone 3

Due Date: March 25th @ 5:30pm

# Class Diagram of Actual System

Facebook chat IM (FBCIM) is a free and open source desktop instant messenger (IM) application that interfaces with Facebook and provides a means for chatting with Facebook contacts. The application is composed of 69 classes. Figure 1 identifies the essential elements of the Facebook Chat Instant Messenger (FBCIM) application.



Figure 1 FBCIM Abbreviated Class Diagram

The application entry point is found in FBCIMApp. This object launches the application and creates the FBChatContext object. The core of the application lies in FBChatContext.

Only one instance of FBChatContext is instantiated. FBChatContext uses FBConnection and FBLoginManager to establish a connection to Facebook and retrieve its contacts list. FBChatContext then uses the services of FBChatManager for the creation of chats.

A chat, or FBChat, created by the FBChatManager is always paired up with an FBContact. A chat has a reference to its related contact. Whenever FBChatContext recieves a chat object from the FBChatManager, it uses an FBContactList to manage its chat and contact pairs.

## Comparison of Domain Model (DM) to Class Diagram (CD)

Figure 2 Domain Model and Class Diagram

Similarities and Differences

There are several common elements between the two models. The CD:FBContact is equivalent to the DM:Friend element. Both models make use of Login services and Connection/ Communication and associated properties. The primary topic of the application, the Chat, is also represented in both models. The CD:FBContactList finds its equivalent in the DM:FacebookAccount’s association to DM:Friends.

The differences manifest themselves in various ways. The Facebook domain elements such as the DM:Wall are not found in the CD. These elements are accessed by the FBChatContext through the connection to Facebook and this function is not extracted to a specialized component.

The CD makes use of Pure Fabrications, not found in the DM, such as the FBChatManager for the construction of chats.

Another subtle difference is in the use of contacts. The DM classified all Friends and the Facebook account owner as Facebook Users. The CD however makes a distinction between the two. The owner’s information is maintained within the CD:FBChatContext while friends are identified as CD:FBContact. The owner is not an FBContact.

In terms of Architecture, the differences between the Domain Model and the Class Diagram are likely the result of design decisions made as the project evolved. The Wall and the Postings found in the Domain Model are elements that may fit well conceptually but are difficult to implement in that framework. As such File Transfer and Facebook Status Update are implemented independently. A significant difference between the Domain Model and the Class Diagram is the Class Diagram’s FBChatContext. The FBChatContext is a dominant component of the architecture and overly so. Its disparity from the Domain Model mark it as a prime element for review in analysis and rework of the architecture.

Tools

Enterprise Architect (Sparx Systems) was used for the generation of the overall class diagram (please refer to Class Diagram in Appendix), which proved invaluable for identifying relationships between objects. The tools recognized most relationships but failed to identify links to classes used by generic type declarations. Visio (Microsoft) was used for the less comprehensive diagrams presented in this section. The online tool Gliffy ([www.gliffy.com](http://www.gliffy.com) ) was used for the partial class diagrams (Figure 3 and Figure 4).

Sample Classes

The method and attributes declarations for FBChatContext and FBChatManager are presented below, including a brief discussion of relationships between the two classes. The relationship between the two classes is also illustrated in Figure 1 and Figure 2.

**public class** FBChatContext {

…

**private** FBContactList contactList;

**private** FBChatManager chatManager;

…

//Sets facebook chat connection.

**public** **void** setXmppConnection(XMPPConnection newVal)

**throws** Throwable {

…

**this**.chatManager = **new** FBChatManager(**this**);

contactList.addContactListListener(**this**.chatManager);

…

}

**public** FBChat chat(FBContact targetContact) {

FBChat chat = chatManager.getChat(targetContact);

**return** chat;

}

**public** FBContact getFBContact(String contactId) {

**return** contactList.getFBContact(contactId);

}

**private** **class** SaveChatHistoryTask **extends** TimerTask {

…

@Override

**public** **void** run() {

chatManager.saveChatHistory();

}

}

**public** **class** FBChatManager {

**implements** FBContactListListener {

**private** **final** Map<FBContact, FBChat> chatMap;

**private** FBChatContext context;

**private** FBChatHistoryArchive archive;

**public** FBChatManager(FBChatContext context) {

**this**.context = context;

**this**.archive = **new** FBChatHistoryArchive(context.getArchiveDir());

**this**.chatMap = Collections.*synchronizedMap*(**new** HashMap<FBContact, FBChat>());

}

**public** FBChat getChat(FBContact contact) {

FBChat chat;

**synchronized** (chatMap) {

chat = chatMap.get(contact);

**if** (chat == **null**) {

chat = **new** FBChat(context, contact);

**if** (context.getSettings().getSaveHistory()) {

chat.setHistory(archive.getHistory(contact));

}

chatMap.put(contact, chat);

}

}

**return** chat;

}

**public** **void** saveChatHistory() {

…

**synchronized** (chatMap) {

**for** (FBChat chat : chatMap.values()) {

archive.saveHistory(chat.getContact(), chat.getHistory());

}

}

}

**public** **void** updateGamesBarVisibility() {

**boolean** gamesBarVisible = context.getSettings().getDisplayGamesBar();

**synchronized** (chatMap) {

**for** (FBChat chat : chatMap.values()) {

chat.getChatFrame().setGamesBarVisible(gamesBarVisible);

}

}

}

As evidence by the code sample, the FBChatContext creates the FBChatManager, and each object has a reference to the other.

The FBChatContext manages a list of FBContacts. With its FBContacts, the FBChatContext uses the FBChatManager to get Chats. The FBChatContext also uses the FBChatManager’s service to save chat history.

The FBChatManager needs the FBChatContext for creating / retrieving Chats (getChat() method). This is an interesting twist since it’s the FBChatContext that makes the call to the FBChatManager:getChat() method, and could pass itself as input rather than relying on the FBChatManager private reference to the FBChatContext. The context is also used by FBChatManager for updating the user interface, a functionality that likely should be extracted to a user interface related class.

# Code Smell

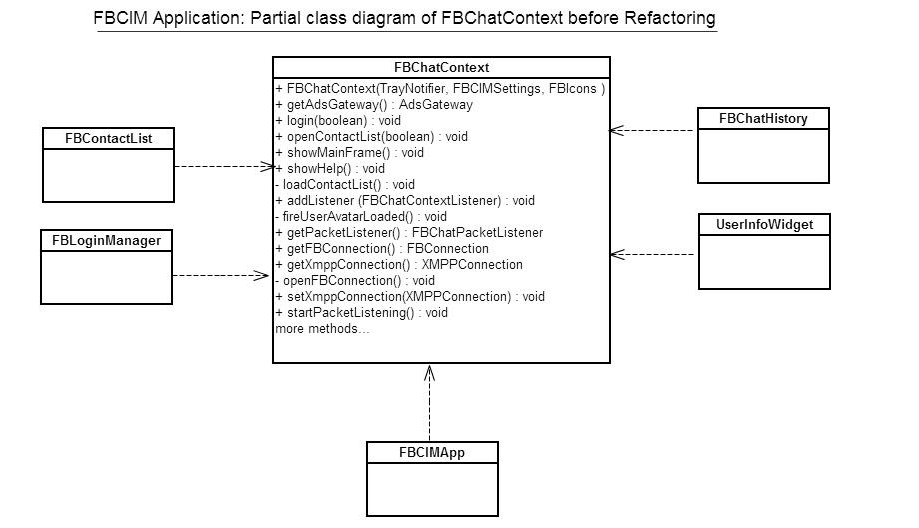


Figure 3: Partial class diagram of FBChatContext before refactoring.

After taking a printout of whole project and collating into a 34” by 33” page, it became clearer that class FBChatContext is doing more than it should. It has been assigned with multiple responsibilities. Responsibilities such as creation of connections, initializing the application context, and tracking and modifying the user status are a few of the many key responsibilities. Multiple responsibilities make FBChatContext less cohesive. In order to fulfill the aforementioned responsibilities, FBChatContext has been coupled with a high number of classes.

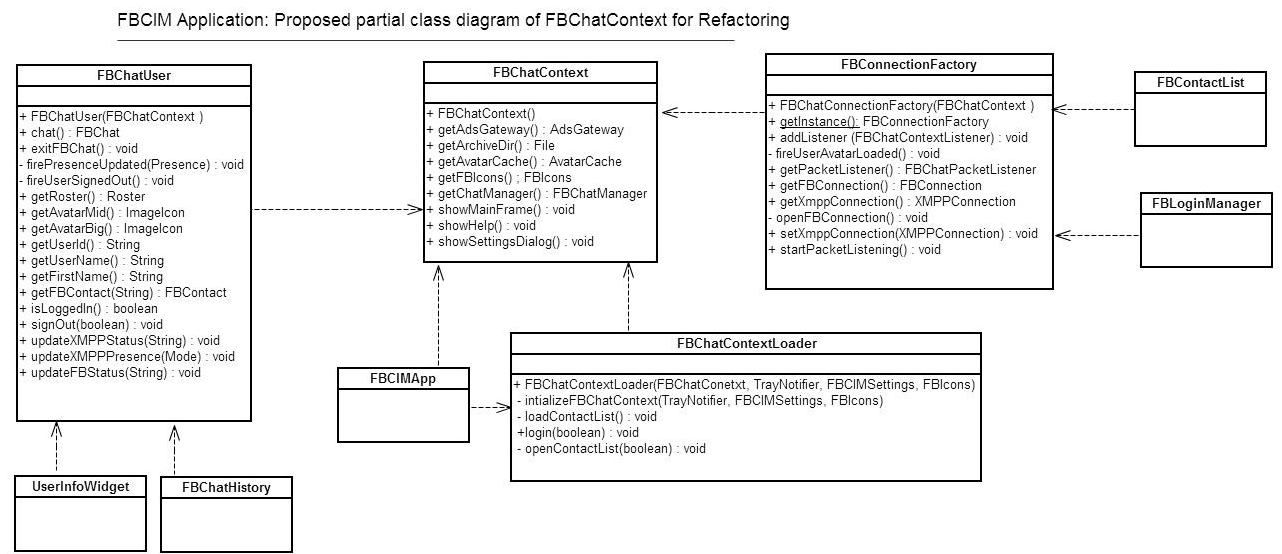
The following are q few noticeable points that make FBChatContext a first choice to re-factor:

1. FBChatContext has a Physical SLOC around 600 and a Logical SLOC around 320.
2. FBChatContext has been coupled to 20 classes (from a total of 69 classes) which accounts for 29% of the software.
3. FBChatContext contains 41 methods (including parameterized constructor) and a few of them are excessively long (e.g. signOut spans 60 lines).
4. FBChatContext has methods to connect with the FaceBook system, to initialize the application context, to track and modify information and status pertaining to currently logged-in user and to provide central access of related information to other objects of the FBCIM application. All these responsibilities make FBChatContext less cohesive and highly coupled class.

**Code Smells** from FBChatContext are as follows.

1. **Large Class:** Aforementioned discussion on more than one responsibility fulfilled by FBChatContext indicates Large Class code smell.
2. **Feature Envy:** Methods named showTrayPopup(FBContact contact)and getAdsGateway()smell of Feature envy as it may makes more sense to have these methods are in class FBContactList.
3. **Long Method:** Methods including (but not limited to) setXmppConnection(XMPPConnection newVal)and signOut(boolean notify)are examples of long methods in the FBChatContext class. It is also important to note that the parameterized constructor of FBChatContext, [FBChatContext(TrayNotifier trayNotifier, FBIcons fbIcons, FBCIMSettings settings)] is a Long Method offender. It creates an inner class which in turn implements a listener interface, implements 3 methods, and use 3 different private classes. In all the FBChatContext constructor spans 40 lines of code for unrelated responsibilities.
4. **Data Clumps:** Fields (instance variables) including (but not limited to) roster, vCard, avatarMid, avatarBig are an examples of data clumps. In an effort to bring higher cohesion to the system, functionality could be refactored around data clumps, regrouping related methods to new classes (making use of the Move Method refactoring technique).

# Refactoring



**Figure 4: Proposed Partial class diagram of FBChatContext for refactoring.**

Our prime focus is to increase cohesion and lower coupling of FBChatContext. We can achieve this by applying refactoring techniques such as **Extract Method**, **Move Method**, **Extract Class** and **Rename Method** to the aforementioned code smells. In applying those techniques, we can move the responsibilities of FBChatContext to 3 new classes, reducing dependencies between FBChatContext and other software classes and thereby lowering coupling. The introduction of 3 new classes will also increase cohesion. We will now describe the new class diagram of the FBCIM architecture (partial) in the context of FBChatContext.

1. **FBChatContextLoader:** FBChatContext class is the core class as it initializes the application context. FBCIMApp calls the parameterized constructor of the FBChatContext class to create its instance. FBChatContext(TrayNotifier trayNotifier, FBIcons fbIcons, FBCIMSettings settings) does more than it should, as already explained in the previous section. We can move that responsibility of initializing FBChatContext to FBChatContextLoader. A step by step explanation of the refactoring techniques required to construct FBChatContextLoader are now presented.
2. Moved the constructor code into new method named initializeFBChatContext()(**Extract Method**).
3. Created a new Class FBChatContextLoader (**Extract Class)**.
4. Copy methods initializeFBChatContact(), loadContactlist(), openContactList(boolean showMainFrame) into the new class FBChatContextLoader (**Move Method)**. It is important to note that 3 private classes also need to be moved from FBChatContext to FBChatContextLoader as they were solely used in the application initialization. Those classes were originally part of the FBChatContext java file.
5. Modify the references in FBCIMApp to initialize FBChatContext via FBChatContextLoader. The following is the code that implements that modification.

**Before Refactoring:** this.context = new FBChatContext(trayNotifier, fbIcons, settings);

**After Refactoring:** ctxIntializer = new FBChatContextLoader(context, trayNotifier, fbIcons, settings);

1. Test the code after building the new jar through build-fbcim.xml file.
2. Remove the following methods from original class FBChatContext initializeFBChatContact(), loadContactlist(), openContactList(boolean showMainFrame).
3. Move method named login(boolean silentLogin)from FBChatContext to FBChatContextLoader to remove **Feature Envy** code smell (**Move Method)**.
4. Test the code once again to test changes mentioned in step g).

Note that class FBChatContextLoader is a candidate for refactoring techniques like Extract method and Move method but this will not be addressed at this time as our primary focus is the FBChatContext class. Also, since FBChatContext class has no dependency on FBChatContextLoader class, refactoring FBChatContextLoader will not benefit FBChatContext.

1. **FBConnectionFactory:** FBChatContext class was also responsible for creating connection objects required for interaction with the Facebook System. Refactoring techniques and the Factory Pattern will allow moving this responsibility to a new class we name FBConnectionFactory. FBLoginManager and FBContactList are coupled with FBChatContext to retrieve XMPPConnection and FBConnection respectively. Generation of FBChatConnectionFactory allows us to move these dependencies from FBChatContext to FBConnectionFactory. This reduces coupling and increases cohesion for FBChatContext. The Refactoring techniques applied to move the mentioned responsibilities to the newly created FBConnectionFactory are explained:
2. Created a new class named FBConnectionFactory(**Extract Class**).
3. Copy named methods including (but not limited to) setXmppConnection(XMPPConnection newVal), FBConnection getFBConnection(),public XMPPConnection getXmppConnection(). into the new class (**Move Method)**.
4. In FBLoginManager and FBContactList, replace references to FBChatContext with FBConnectionFactory.
5. Test the code after building the new jar with build-fbcim.xml file.
6. Remove methods copied from FBChatContext.
7. Test the code for the changes made in step e).
8. **FBChatUser:** The FBChatContext class was also responsible for keeping track of information and status pertaining to logged-in user. Refactoring techniques allow moving this responsibility to a new class, FBChatUser. FBChatHistroy, UserInfoWidget and TrayPopUpMenu are examples of a few classes that have been coupled with FBChatContext to retrieve user information through methods like getUserId(), getUserName() and getFirstName(). Generation of FBChatUser allows us to move these methods (thus dependencies) from FBChatContext to FBChatUser, hence reducing coupling and increasing cohesion for FBChatContext. Refactoring techniques applied are explained below in **step by step** instructions for creating FBChatUser.
9. Creat a new class named FBChatUser(**Extract Class**).
10. Copy methods named above, including but not limited to, getUserId(), getUserName() and getFirstName() into the new class named FBChatUser (**Move Method)**.
11. Change references of FBChatContext to FBConnectionFactory inside FBChatHistroy, UserInfoWidget and TrayPopUpMenu.
12. Test the code after building the new jar with build-fbcim.xml file.
13. Remove code of methods copied from FBChatContext.
14. **FBChatContext:** FBChatContext class now has a relatively low number of methods and more importantly, the remaining fit better with the intent of the class. The remaining methods include public AvatarCache getAvatarCache(), public FBIcons getFBIcons(), public File getArchiveDir(), public FBCIMSettings getSettings(). By their very nature, these methods should be in a class that is central to the application and offer their services available from a single point of contact.

# Source code and Suggested Refactoring

**public** **class** FBCIMApp

**implements** FBChatContextListener {

// ……………………… code removed due to limited space……………………………………

**public** **void** initContext() {

**this**.context = **new** FBChatContext(trayNotifier, fbIcons, settings);

**this**.context.addListener(**this**);

}

}

**public** **class** FBChatContext {

**public** FBChatContext(TrayNotifier trayNotifier, FBIcons fbIcons, FBCIMSettings settings) {

**this**.listeners = Collections.*synchronizedSet*(**new** HashSet<FBChatContextListener>());

**this**.trayNotifier = trayNotifier;

**this**.fbIcons = fbIcons;

**this**.settings = settings;

**this**.trayNotifier.setContext(**this**);

// ……………………… code removed due to limited space……………………………………

**this**.fbChatMainFrame = **new** FBChatMainFrame(**this**);

**this**.fbLoginManager = **new** FBLoginManager(**this**, **new** FBLoginManagerListener() {

**public** **void** loginSuccessful(**boolean** showMainFrame) {

openContactList(showMainFrame);

loadContactList();

}

// ……………………… code removed due to limited space……………………………………

**public** **void** loginCanceled() {

**new** MessageDialog(**null**, "Error", "Login to Facebook chat canceled!", "OK").showMessageDialog();

signOut(**true**);

}

}

**public** **void** login(**boolean** silentLogin) {

fbLoginManager.setSilentLogin(silentLogin);

fbLoginManager.login();

}

**public** **void** openContactList(**boolean** showMainFrame) {

// ……………………… code removed due to limited space……………………………………

}

**private** **void** loadContactList() {

// ……………………… code removed due to limited space……………………………………

**this**.serviceTimer.schedule(**new** SaveChatHistoryTask(),*HISTORY\_SAVE\_INTERVAL*,*STORY\_SAVE\_INTERVAL*);

**this**.serviceTimer.schedule(**new** UpdateAdsTask(), *UPDATE\_ADS\_INTERVAL*, *UPDATE\_ADS\_INTERVAL*);

}

}

}

**private** **class** UpdateFBStatusTask **extends** TimerTask {

**public** **void** run() {

contactList.updateContactsFBStatus(skipOffline);

**if** (!skipOffline) {

skipOffline = **true**;

}

}

}

Note: Apart from FBCIMApp and FBChatContext there are 3 other private classes that are being instantiated and referred solely in constructor method. Only one private class has been shown due to limited space.

**Explanation on Suggested Refactoring**

The FBChatContext class is the core class as it initializes the application context. FBCIMApp calls the parameterized constructor of FBChatContext class to create its instance. FBChatContext(TrayNotifier trayNotifier, FBIcons fbIcons, FBCIMSettings settings) does more than it should as already explained in the previous section. We can move the responsibility of initializing **FBChatContext** to **FBChatContextLoader**. Continuing the refactoring effort and moving additional responsibilities out of **FBChatContext** to **FBConnectionFactory** and **FBChatUser** further improves the application’s architecture.

# Appendices

1. Java files for FBCIMApp and FBChatContext.





2. Class Diagram generated using Enterprise Architect

