Project Description

Messenger services or applications are the heart of internet communication. Android is a new platform for communication which is mobile, energy constraint and application rich. With the intention of working on some killer app using distributed coordination and communication technology we have chosen to develop a Messenger Service using native Android API. Android Messenger is a Java application runs on Android devices or Emulators to communicate with each other with the help of centralized server. The centralized server implements Tuple Space based co-ordination system which enables the message communication among the Android devices. TCP/IP implemented with Java Socket API has been used for all network level communications.

The Four main component of the Android Messengers are described below,

Android Client: It is a Java application developed using android API. Application is developed in using Android Eclipse plugin and emulator. The source code of the client application contains code for user interface, Protocol for communication and the threads for sending and receiving tuple requests.

1. **User Interface:** The simple user interface of Android Messenger consists of EditBox to write message, Button to send message and Array Adapter to display typed or incoming messages. OnClickListener used as an event listener of types message which call method of MessageCommunication to forward the message to tuple Space.

2. **MessageCommunication:** The MessageCommunication java class creates message object and routes the request to appropriate thread of communication.

3. **Threads:** Receiver and RequestTuple threads remain alive all the time with Android client to listen the incoming requests or to send Tuple requests to server.

Tuple Server: TupleSpaceServer listen on incoming request with the help of SessionServerThread. SessionServerThread forward the request to TupleSpaceServer by calling processRequest method. The processRequest method checks message object field and process accordingly. The operation field of message object contains read, write or take request. This server which also manages tuple space calls appropriate method on tuple space to read/write tuple. Messages are converted into tuple by the server and vice-versa from tuple to message object.

Tuple Space: The tuple space is implemented using Java collection class ArrayList<Object[]> i.e. array list of Object array. The read, write and take operations are broken in to get, add and remove method of arraylist. Tuple matching has been done by iterating over arraylist and first successful matched return as result. Tuple space containing duplicate values returns randomly any of the duplicate value but our tuple space return first match value.

Message: Pre defined message class on client and server machine should be available to instantiate the serialized object. The messages objects are serialized to string by XStream api and vice-versa deserialized from string to Message object. Message class contains following fields

1. **sender:** Contains sender’s id.
2. **receiver:** Contains receiver’s id.
3. **operation:** Nature of operation expected on TupleSpace, i.e. read, write and take.
4. **message:** Original string message requested by the messenger or to displayed on the messenger.
**Research Paper Analysis #1**

**Paper Name:**
A Distributed Hash Table based Tuple Space Service for Distributed Coordination

**Author:**
Yi Jiang, Guangtao Xue, Zhaoqing Jia and Jinyuan You

**Analysis:**
This paper deals with the implementation of distributed tuple. Peer to Peer hash table based architecture has been used to distribute Tuple space over network. This paper introduced publisher/subscriber model for Tuple Space which we have used in our project. Requesting entity subscribes the message from publishing entity. An entity (server or client) could take the role of publisher or subscriber based on application flow. More advanced feature discussed in this papers are agent based control and coordination among distributed tuple spaces. Tuple spaces are further divided in public and private domain i.e. public space is available for external agents. Inter tuple space communications are suggested as transaction based communication due to atomicity of the operation performed on this tuple space.

**Research Paper Analysis #2**

**Paper Name:**
A Tuple Space for XML & its application in Orchestration of Web services

**Author:**
Umesh Bellur, Siddharth Bondre

**Analysis:**
The paper provides an implementation of system independent Tuple Space architecture. Tuple space as webservice implies tuples are XML file, reading tuple is reading xml and writing is updating or creation of new xml files. XML files are architecture and protocol independent entity which could be transferred over network. Sterilization and deserialization is quite easy on xml file due to its predefined format and readily available parsers. In our project, we have used xml like message string for the communication. This paper discussed java based implementation though due to its generic nature it could be ported in any environment. The implemented components are web clients, MBeans, xSpace storage group.
Research Paper Analysis #3

Paper Name: A Study of Internet Instant Messaging and Chat Protocols

Author: Raymond B. Jennings III, Erich M. Nahum, David P. Olshefski, Debanjan Saha, Zon-Yin Shae, and Chris Waters, IBM T.J. Watson Research Center

Analysis:
Instant messaging (IM) and network chat communication have seen an enormous rise in popularity over the last several years. However, since many of these systems are proprietary, little has been described about the network technology behind them. This paper helps bridge this gap by providing an overview of the available features, functions, system architectures, and protocol specifications of the three most popular network IM protocols: AOL Instant Messenger, Yahoo! Messenger, and Microsoft Messenger. It describes common features across these systems and highlight distinctions between them. It also discusses the advantages and disadvantages of different technical approaches used in these systems to support different features and functions. Then it briefly discusses the ongoing efforts to standardize IM and chat-based protocols in IETF and other standards bodies.

Project Design

1. Flow Diagram:
2. Class Diagram:

[Diagram of class diagram showing relationships between classes such as Android Messenger, RequestTuple, SessionServerThread, Receiver, MessageCommunication, Receiver, Message, TupleSpaceServer, TupleSpace, and TupleSpace.]
Installation Manual

Server Installation Guide:

1. Unzip the AndroidMessenger.zip file
2. Open the “AndroidServer” Folder thus extracted.
3. Run the TupleSpaceServer.java file
4. The output will show the server IP address.
Client Installation Guide:

Android device 1

1. Go back.
2. Go to the “Android Messenger Client\src\com\vg\am” folder, open the AndroidMessenger.java
3. at line number 78, update the `serverAddress` string with the IP address from the point 4 server installation guide.

4. Go to line 56 and edit with `thisClient <android client 1 name>`
5. Go to line 57 and edit with `requestClient <android client 2 name>`
7. Dispatch Run the client on the android client 1.

Dispatching on Emulator:
Android device 2
1. Go back.
2. Go to the “Android Messenger Client\src\com\vg\am” folder, open the AndroidMessenger.java
3. at line number 78, update the \serverAddress\ string with the IP address from the point 4 of server installation guide.
4. Go to line 56 and edit with \thisClient <android client 2 name>
5. Go to line 57 and edit with \requestClient <android client 1 name>
7. Dispatch Run the client on the android client 1.

Dispatching on Emulator:

Note: You have to make sure that the server and both the clients are working on the same network. (if the network is a Wi-Fi network then switch on the Wi-Fi on both the devices and connect them to the same network)
**User Manual**

User needs to make sure that the Wi-Fi of the device is switched on and connect to the same network to which the server is hosting the chat service and switch on the software. If everything works fine then the two android devices are connected to each other and a chat screen pops up where in the users can chat with each other.

Once the program is running type the message to be sent and press the Send button.

Example of the project running on an emulator:
**Project Result**

1. A distributed messenger service for Android devices using android API.
2. Tuple Space implementation on Java virtual machine.
3. TCP/IP socket communication among Android client on Dalvik virtual machine and Tuple space server over Java virtual machine.
4. A new object serialization technique for Android which uses XStream API.

**What we learnt?**

1. Better understanding implementation and development of tuple space.
2. Better understanding of implementation of tuples.
3. Learned the Android API and programming paradigms on Android platform.
4. Learned to deal with android specific programming and implementation issues while programming on Android.
5. Debugging Android errors using Log class.
6. Though didn’t implement but we have learned Google App Engines API.
7. Learned to develop Socket based communication system on android devices.
8. We studied Professor Alan Kaminsky’s API for Simple Space but we didn’t implement it due its tight coupling with registry server.

**Possible Future Work**

1. To implement this Android messenger service over Wide Area Network we need to implement this tuple space server on Google app engine or any web server.
2. We need to transform all socket communication to http communication.
3. Rich user interface with more features such as authentication, multiple chat window, conferencing, file sharing etc.
Member Contribution

Gourav Mitra:
- Researched and Studied various research papers on the tuple space, DTuple Space, XSpace and their working.
- Developed custom tuple space on the server.
- Designed the server.
- Developed the Server side of the software.
- Worked on deliverables.

Varun Goyal:
- Researched and Studied various research papers on various Instant Messaging services.
- Studied Android API
- Designed the client side.
- Developed the Client side of the software.
- Developed the connection mechanism between the server and client.
- Studied Google App API
- Created web server on Google Apps http://illuminati-rit.appspot.com/
- Worked on deliverables.
- Created and maintained the team web-site.

Nipun Sud (Dropped the subject):
- Researched and studied various research papers on tuple Space

Work that was expected from Nipun
- Dynamically pass the server address to the client.
- Setup the server on Google Apps web server.
- Create the welcome screen for client to provide client credentials.
- Search for a way to provide device ID dynamically to the server for client verification.