# OO Based Development of a Multi Media Application Server Prototype

E. GUL, G. WILLEKENS(team leader), F.HOSTE, T. BATSELE, R. SELDERSLAGHS, N. QUARTIER
Alcatel Bell (A7)
Francis Wellesplein 1
2018 Antwerpen, BELGIUM

*Abstract:-* This paper presents how a complex communication application server and its clients have been developed using OO techniques and programming environment. Multi Media Application server (MMAS) provides voice, video and data services for IP clients. The programming language Java was chosen to implement the overall system, because it is platform independent, supports object serialization, network programming, multi threading and provides an AWT for graphical user interface design. The clients and the server communicate using sockets, there is a CORBA interface between MMAS and its repository (database). The client can be run as a Java application or with slight modifications as a Java Applet.

*Key-Words*: - Object oriented design and programming, Multimedia, Communication systems, Java

## 1. Introduction

OO Design and programming addresses the issues of designing complex programs. Even though OO Design has been used in many successful projects, designers are usually not comfortable in applying these concepts on different domains, i.e. communications. The aim of this study is twofold, the first one is to design a Multi Media Application Server prototype using pure OO Programming Environment, and the second one is to show how a communication system can be designed and implemented in this environment.

In MMAS there are many modules ranging from communication protocol implementation to GUI design. The only language, which can be used for all of these modules, is Java. Besides its platform independence, it supports multi-threading, object serialization, socket programming, and has an AWT for GUI design.

MMAS can be thought as a controller who sits on signaling layer. Signaling layer is for making connections. It does not deal with actual voice or video communication. In signaling layer SIP stack was used. [1]. To transport voice and Video Real Time Protocol provided with Java Media Framework was used [2]. The block diagram of MMAS, SIP stack, JMF (RTP streams) and WEB Server is shown in Figure 1. Web Server hosts web pages of MMAS, registration applet and servlets to configure supplementary services. the next section server side of MMAS is explained. In section 3 MMAS clients are described. Finally discussions and conclusions are in section 4.

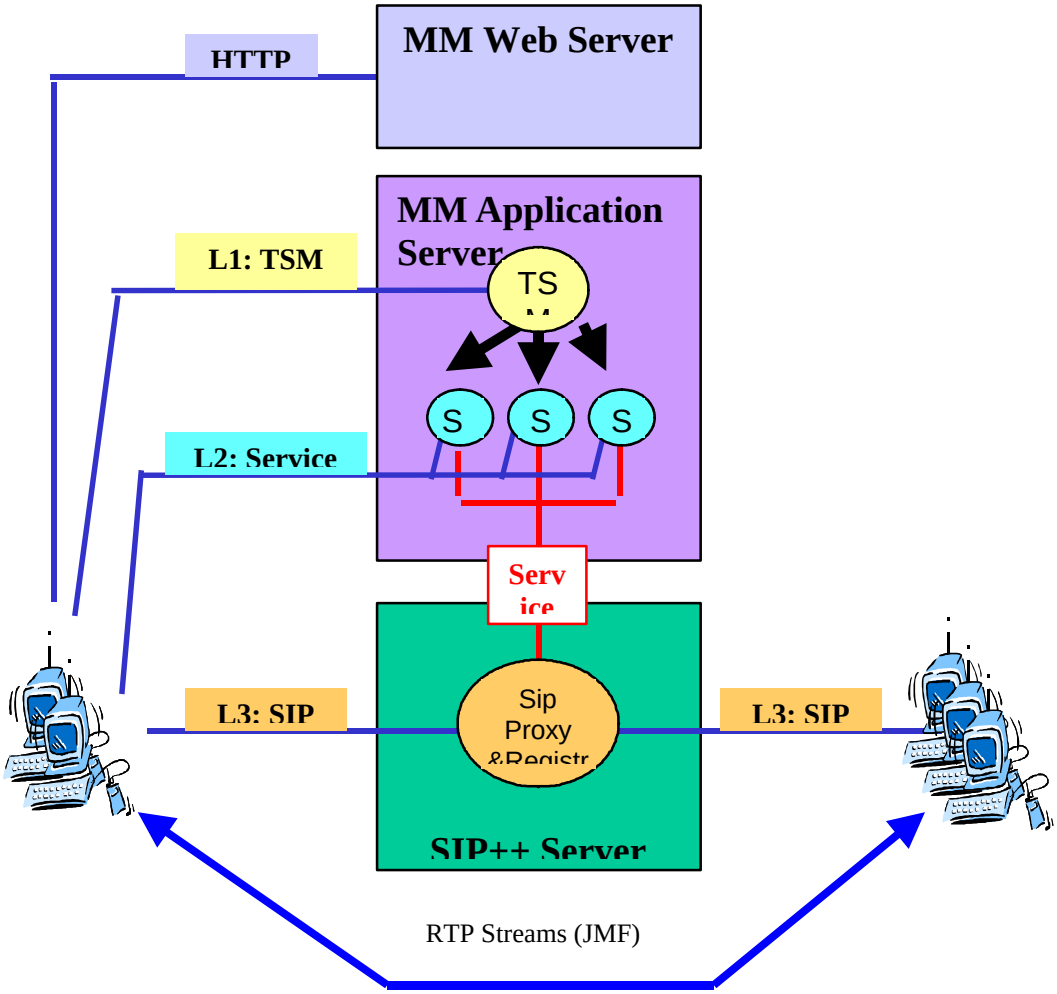


Figure 1. The Block Diagram of MMAS, WEB Server, SIP Stack and Media Streams

## 2. Server Architecture

The architecture of Multi Media Application Server (MMAS) is shown in Figure 2. The telecom service factory is the main component of the system. It creates Telecom Service Managers (TSM) which do the actual processing, creates service factories objects, which are responsible for supplementary services, and initializes ORB. It also keeps track of all created TSM instances. TSM Factory is singleton object. There can be only one instance of TSM factory in MMAS.

The telecom service factory has server socket which listens incoming connections in an infinite loop. Whenever a connection is received from a client, a TSM object is created and socket is passed to TSM. TSM is a thread object, when it completes its task, it dies.

When TSM deals with a service request, it retrieves the appropriate service factory from the service factories table. For example, we assume that this service is “park and pick service”. There may be many users who are using this service. Park and Pick Service Factory returns the instance of the park pick service for this user. If the service is used for the first time, then new service object is created.

Once created service objects are kept alive forever, i.e. until MMAS is terminated. Service objects depending on the type of service may communicate with clients and SIP stack.

The users need to register to MMAS to use the clients. Registration information is collected by an applet which will be explained in the next section. The applet send a request to TSM Factory and a TSM instance is created for registration. For registration, the user database should be interrogated first to know if the user is already in the database. TSM invokes a method of Repository object. The repository object interrogates the database and returns the result. If the user is not in database, then the user information received from the applet is sent to the database by calling the methods of repository object. The information received from applet is a serialised object, which contains the user attributes such as username, password, name, picture and logo. The repository object is located on another machine (database server). The communication between TSM and Repository is via ORB provided with JDK. This ORB does not support Object by value method calls; therefore the user and picture are converted into byte arrays before transferring them to the database server.

In MMAS both TSM and Repository were implemented in Java. In this case, it is possible to use Java RMI between TSM's and Repository. This was also implemented. However in actual product, database server and its interfaces could be in another language, therefore CORBA middleware is more appropriate.

On the user terminal GUI, the picture of the user and called user are displayed. The terminals request these pictures from TSW Factory. TSW Factory creates a TSM. This TSM passes the get picture requests to the Repository. The repository retrieves the pictures and sent them back to the TSM as byte arrays via ORB. These byte arrays are converted into serialised objects and passed to clients to display them on GUI.

When the user logs on, the username and password are sent to TSW Factory. TSW Factory creates a TSM to authenticate the user. TSM calls a method of Repository. This method finds the username in the repository and checks if the password is correct. It returns authenticated or not authenticated to the TSM. TSM sends this result back to the client.

As mentioned above users register to MMAS using an applet. This applet and related web pages were placed on a web server. This web server may run the same machine where MMAS running. However, taking into account of the performance issues the web server should be installed on a different machine. When the user goes to MMAS web page, he can click a link to download register applet to his local machine. Once the browser on the local machine is running the applet, he can enter the required fields.

It also possible to configure some supplementary services using servlets. In this case user clicks configure services link and enters his username and password. A servlet gets this information and initiates a session for this user. The username and password are sent to MMAS for authentication using the socket connection. TSW factory creates a TSM. This TSM invokes authentication method of the repository. After the authentication, another servlet displays a configuration page to configure services. The user selects a service and configures it. Configuration data is handled by servlets again and passed to TSM via socket connections.

Configuration of some services such as call forwarding can also be done by MMAS terminal. TSM does not know whether the configuration requests coming from servlets or MMAS terminal. TSM analysis the command and returns the necessary information. If the servlets make the requests, the information received from TSM is used to prepare an HTML page. This HTML page is sent to the browser. On the other hand if MMAS terminal makes the request, the information received will be displayed in a pop up window.

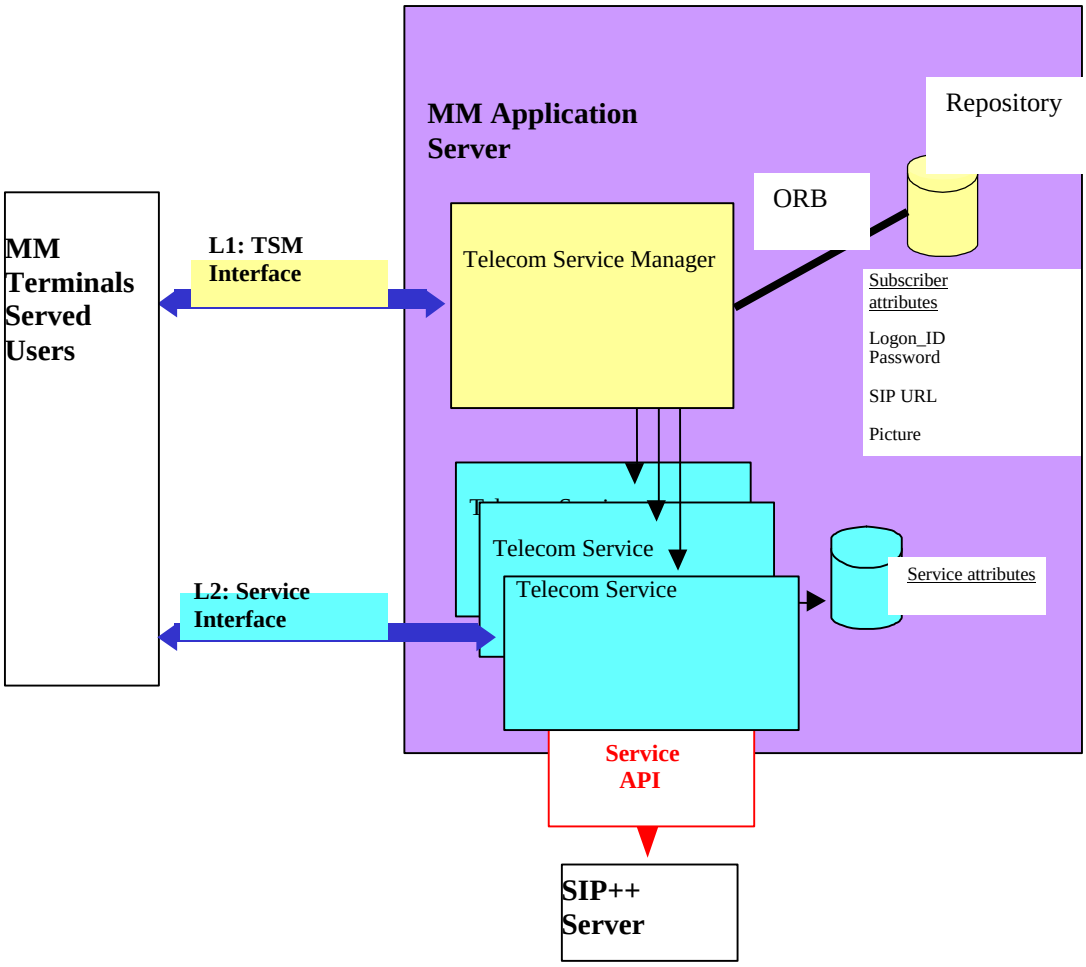


Figure 2. The Architecture of MMAS

## 2. Clients

The client of MMAS is Multi Media Terminal. The Multi Media Terminal is a Java application. It has a GUI to select applications such as Internet Phone, Mail, and Video. For the moment, only the application "Internet Phone" has been implemented. Internet phone is based on SIP stack and supports audio, video and data communications.

The top layer is called Telecom Service Wizard (TSW). It acts as a portal that allows access to the underlying applications. It is foreseen that several (possibly all) applications may have to make use of SIP for their remote communication. This is exactly the purpose of the third layer, represented by the "SIP Phone" package, which on the one hand offers a standard (SIP) interface to the applications that wish to make use of SIP, and on the other hand, distributes incoming SIP requests to the appropriate application.

SIP stack is also in Java and has a layered architecture. It has four layers. SIP Phone has an interface with third and fourth layer only. The architecture of this SIP stack is discussed in [3].

Internet Phone deals mainly with three aspects, being:

- Graphical User Interface,
- SIP-call and SIP-call-leg (dealing with the "signaling" aspects) ,

- Connection-call and Connection-call-leg (dealing with the “bearer” aspects).

The major classes of Internet Phone are described below.

CallManagerGUI handles all actions towards the graphical components. It hides the graphical implementation (at least the details of it) from the outside world (rest of the system). It is split in a part for incoming calls, outgoing calls and active calls. Also there is a part that manipulates the “me” icon i.e. the icon that represents the served user. This class is the view of the Call Manager which acts as controller and implements the model

The Call Manager is the “central class” of the Internet Phone. Its tasks can be summarized as follows:Be the “focal point”,

a) between the served user and the “call handling” in the terminal,

i) Convert the call handling related commands that the served user issues via the GUI, into specific (but still rather high level) commands towards one the one hand the SIP stack (the signaling plane) and the realm of the “streams” (the bearer or connection plane).

ii) Convert the requests of the remote users (coming in via SIP) towards indications on the GUI allowing the served user to react on them.
To be able to perform this task, it is clear that the call manager needs an overview over all calls from or towards the served user.

b) between the served user and the “service logic” in the MM server.

It also terminates the L1 and L2 communication towards the MM server. Via these two communication layers, the user can issue service related commands towards the applications. In the MM server, and vice versa, applications in the MM server can push information that is meant for the served user, on the GUI.

The MMAS terminal is a Java application, however it has also been converted to Java applet. The code was developed using jdk1.2. For the time being, no browser supports applets which use jdk1.2. Therefore, it is necessary to load Java1.2 plugin to run the applet

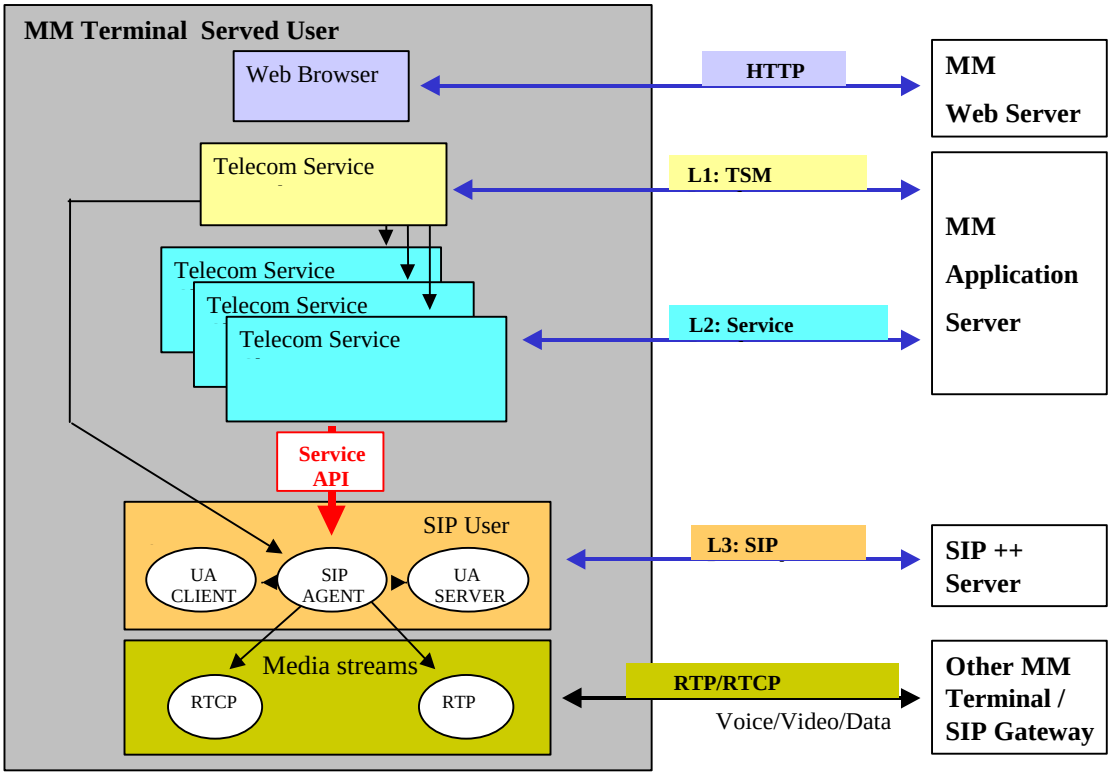


Figure 3 MMAS Terminal

An applet, which is called register applet, was developed to register new users to MMAS. This applet downloaded to user terminal by clicking its link on MMAS web page. It requires Java plugin to run. The user chooses an username and password, he also enters name, surname and URI. The logo and photo of the user are read from the local hard disk. All information is stored in an object called Subscriber. This object is serializable and sent to MMAS via TCP sockets. Since accessing the local disk and making socket connections breaches applet security mechanism, we need to prepare a policy file to give read access and open socket connection rights to the applet. This policy file should downloaded and stored in the user home directory.

## 4. Results and Conclusions

A Multi Media Application Server Prototype has been developed using OO Design and programming. All modules of this prototype are in Java. The communication mechanism between MMAS and its clients is via sockets. However, inside MMAS Corba was used. It has been considered to use Corba between terminals and the server. In this case, the design of communication mechanism between clients and server would be easier, however the performance would had been lower. Therefore, it was decided to use socket mechanism in client –server communication.

The MMAS creates a thread for each request. If number of requests is more than a certain value, no more threads could be created. The server gives thread panic exception. In the real product, incoming requests should be hold in a queue, and a pool of threads should be created. Then a thread from the pool should be assigned for a request.

Even though Java supports threads, there is no guarantee that an event will be executed in a given time. The current Java implementation does not support real time events. There is a proposal for real time Java implementation. Until real time Java becomes available, it will not be advisable to use Java in a carrier grade communications system. However, it can be used in simple clients.

The concept and design of applets are attractive, however in real life is not easy to write and deploys applets. Current browsers support only old version of jdk. The applets developed in jdk1.2 need a Java plugin. In controlled environments, like corporate LAN's, applets can be deployed.

Sound and video can be integrated in Java applications and applets using JMF. However, JMF is not stable yet. If this product becomes mature, then it can only be considered for real product development

OO Programming Environment, particularly Java is well suited to develop prototypes. However, for real products, because of the performance issues and accumulated experience, procedural languages, especially C is the choice.

*References:*

[1] M. Handley, H. Schulzrinne, E. Schooler and J. Rosenberg , *SIP: Session Initiation Protocol,*: IETF-RFC 2543, March 1999.

[2] Java Media Framework(JMF) API, http://java.sun.com/products/java-media/jmf/

[3] D. Chantrain and N. Marley , H. Zou, H. Wang, W. Mao, B. Wang, S. Focant, K. Handekyen, *Prototyping SIP-based VOIP Services in Java*, IFIP -World Computer Congress-International Conference on Computer and Telecommunications, Beijing, China, August 2000.