Reuse-Oriented Architecture of an Integrated Multimedia Messaging Application

MIHAI DINSOREANU
iQuest Technologies
Motilor St. 6-8,
40001 Cluj Napoca
ROMANIA

IOSIF IGNAT
Computer Science Department
Technical University of Cluj-Napoca
Baritiu St. 26-28
ROMANIA

Abstract: - Integrating different means of communication is one of the most actual and challenging issues nowadays. In addition to that, an integration system should also be constructed using an open architecture, allowing for adding new services based on different technologies. This paper comes with its own contribution to this global goal by introducing a novel approach meant to bring simplicity and interoperability to and among various communication technologies. The presented architecture is reuse-oriented, integrating suitable architectural patterns allowing for the development of different application systems according to the specific requirements of the communication providers (i.e. Telecom operators or ISPs).

Key-Words: - Reuse, architectural patterns, multimedia messaging.

1 Introduction

Due to the significant number of communication means available these days the need of an integrated, single communication point becomes increasingly important. An integrated communication point should provide functionalities like: e-mail, instant messaging, VoIP, conferencing, and also allow for adding new services like online storage and application sharing or secure communication over networks opened by third-party operators. The aforementioned services are based on different technologies and although we can mention different approaches for bridging those technologies, they are still far from defining a standard. One standardisation effort is represented by SIP, the Session Initiation Protocol, a signalling protocol for Internet conferencing, telephony, presence, events notification and instant messaging. SIP was developed within the IETF MMUSIC (Multiparty Multimedia Session Control) working group, with work proceeding since September 1999 in the IETF SIP working group [1]. SIP is a method-based protocol stemming from HTTP which gets a step forward since SIP does not explicitly publish specific services but allows the developer to implement them based on a set of well-defined methods. The efforts of the community towards a standard based on SIP is reflected in an increasing number of SIP-enabled devices on the market, considering therefore SIP as the underlying protocol of our framework makes sense. Integrating the above mentioned functionalities and services is possible if one considers using SIP as the protocol for user presence signalling and authentication. However, the framework should allow connecting not only SIP clients but also instant messaging clients (i.e. H.323) or classical telephony clients (i.e. PSTN).

This paper presents a framework that allows creating application systems customized according to the telecom operator or Internet Service Provider requirements and specifications. The framework, called Integrated Multimedia Messaging (IMM) supports functionalities like:

- Secure access to services both wired and WiFi
- Voice and Video transmission
- Coupling points for external gateways linking the SIP-based network to PBX-based networks for corporate networks
• Coupling points for already existing private PSTN clients
• Connecting IP phone SIP-enabled clients
• Voice Messaging
• Audio/video conferencing
• Instant Messaging
• Internet Phone programmable services like voice mail

In order to provide the above-mentioned functionalities and to be flexible enough to allow for specific applications, the framework architecture is a multi-tier architecture based on several architectural patterns as presented in the following sections.

2 Functional Description
The required functionalities of the system are described in the following sections

2.1 Use-case Model for IMM
In this paper, we consider the three main types of actors: System Administrators (SA), Desktop Users (DU) and Mobile Users (MU). The abstract user concept is the Authenticated User (AU) factoring out the common properties of DU and MU. AU is the main user of an IMM-based application via a client application and we present next the functionalities of the framework related to AU. The IMM-based application provides to AU services depending on the current configuration, like: instant messaging, online video streaming, SoftPhone, secure communication etc. Mobile users can access similar services. The architecturally most important use cases are shown in the next use-case diagram in Fig. 1.

![Fig. 1. AU Use-Case Diagram](image)

2.2 Service Description
In the development of our multimedia messaging system, several protocols interact, this protocol system being often referred as the “protocol zoo”. We need:
• Signalling protocol to establish presence, locate users, set up, modify, and tear down sessions
• Media Transport Protocols for transmission of packetized audio/video
• Supporting Protocols Gateway Location, QoS, interdomain AAA*, address translation, IP, etc.

As the signalling protocol we based our system on SIP. The Session Initialisation Protocol (SIP) is an end-to-end, client-server session signalling protocol that resembles more HTTP than to legacy signalling both in terms of service model and protocol design. From the user’s perspective SIP primarily provides IP presence and mobility, while from the developer’s perspective it provides a set of primitives: session setup, termination, changes. Starting from these primitives, SIP messages can convey arbitrary signalling payloads and therefore permit the development of arbitrary services on top of it, like redirect calls from unknown callers to secretary, reply with a webpage if unavailable or send a jpeg picture on an invitation, and of course, various multimedia services. Applications may leverage SIP infrastructure and therefore SIP is not limited to Internet telephony and is suitable for applications having a notion of session like:
• Distributed virtual reality systems
• Network games (Quake II/III implementations)
• Video conferencing, multimedia and integrated messaging.

For a better understanding of the SIP-based communication model between two IMM users, a typical messaging usage scenario is presented next as a sequence diagram in Fig. 2.

The collaborations presented above are abstract, they are not dependent on the media type involved in the communication process. Let’s consider two instances of AU, John and Jane, and a connection initiated by John from his own PC to Jane which is on an unknown location. John, registered as john@home.com, initiates a dialog (SIP-based unified messaging session) with Jane at address jane@office.com(1). The office.com server answers the SIP agent on John’s machine that user Jane is not available (2). The SIP agent on John’s machine looks up Jane’s configured contacts and tries to contact Jane on the location having the highest address priority, which is the phone on Jane’s table connected to school.edu (3). The school.edu server tries to call Jane’s phone (4) and if she doesn’t
If a timeout message is received from the preferred connection, the school.edu server starts contacting in parallel the other locations configured by Jane, like the university server (6) or any other possible configured contact points, fixed or mobile. The university server (cs.univ.edu) notifies the client application on Jane’s machine that she has a message sent by John (9) and Jane accepts the message (10). cs.univ.edu server sends back the acceptance confirmation to the SIP server, school.edu (11) which in turn suspends all the other attempts to contact Jane (12) and sends an acceptance confirmation back to the client application on John’s machine (13). From this point, the clients on both machines will establish a peer-to-peer connection in order to continue the communication session regardless of its nature.

We present in the next section the general architecture of the IMM framework based on the classical multi-layer architecture.

3 IMM Architecture

Because SIP decouples signalling from transport it makes it service independent and opens the possibility to accommodate an unknown range of services. In this context the architectural goal when developing this system was to base it on principles that will enable in the future its extension with additional functionality.

For the structuring of the system we use the Layers pattern [6], which is applicable to large (distributed) systems that require decomposition and enables us to split tasks in subtasks on different layers of abstraction.

After we shall identify the subsystems and general modules we’ll distribute them across layers having the responsibilities to:

- Layer N provides services used by layer N+1
- Layer N delegates subtasks to layer N-1

The general modules of the IMM-based applications are presented in Fig. 3.

3.1 The Application Layer

The main components of the architecture are:

- The IMM server system containing the User Services server, the Database server, the Proxy server and support servers.
- Client applications like multimedia streaming, messaging or other softphone SIP enabled applications
- IP-enabled phones; client devices that implement SIP protocol can be directly connected to IMM.
- Gateway between SIP systems and PSTN systems
- PBX Telephony Switch
- Individual phones connected to the SIP/PSTN gateway
- Corporate phones connected to the PBX switch

The modules presented above are layered on the classical architectural layers: Application layer, Domain layer, Middleware layer and System layer.
The SIP Server is the central point of the application system and provides:
- A server side implementation of the SIP protocol stack
- Registration services
- Routing services; receives location information from client applications and forwards user messages to other registered users.

The Database Server uses MS SQL and its responsibilities are to store user information, network addresses, phone numbers, user profiles, voice messages etc.

The SIP/PSTN Gateway is a bridge device that should contain an implementation of SIP on one end and an implementation of an IP/PBX bridge on the other end.

The Media Server provides media streaming services, storage and delivery of voice messages and acts as a gateway for media playing client applications like Quicktime or MediaPlayer.

The Unified Messaging Server uses the Media Server and acts as a phone robot and a voicemail server.

The Conference Server is the central server for audio/video conferencing.

The Translator Server SIP/H.323 responsibilities are to bridge from SIP to H.323 and the gateway for traditional messaging clients like NetMeeting clients.

The Web Server hosts two applications: the configuration and management application for the IMM system, users and services and the client configuration application.

The Client applications are based on a layered architecture as well, as depicted in Fig. 5.

The Presentation layer contains the classes related to the user interface and the external I/O devices interface (video camera, microphone).

The Domain layer contains instances of the components related to control, signaling and multimedia functions.

The Middleware layer contains the packages that implement the signalling protocol stack H.255.0 for H.223 and SIP support and the signalling protocol Q.931 on ISDN.

The System layer contains the components for implementing the TCP and UDP stacks and Ethernet packaging.

### 3.2 The Domain Layer

This layer includes specific components packages for the multimedia messaging system like multimedia components – used for multimedia processing both on the server-side (Multimedia Server) and on the client side for image/voice gathering and streaming. These components are:

- Codecs and streaming components
- Conference components – used by the Conference Server
- Messaging Components – used for implementing the classical H.323 messaging functions.
- Session and Signaling components – SDP and SIP.

### 3.3 The Middleware Layer

This layer contains support systems like the .NET Framework, MS SQL and the IIS Server.
3.4 The System Layer
This layer contains the components implementing the network protocols supporting IMM including SIP Stack, TCP Stack or any other needed protocol e.g. CHAP.

4 Using Architectural Patterns
For defining the architectural mechanisms, we consider the architectural components (patterns) similar to software components. First, a set of appropriate patterns are identified and then their interfaces and internal structure is described. Since architectural pattern interfaces are needed for composition purposes, the pattern should not be considered as a black box, the internal structure of the architectural pattern is described as UML class diagrams. Buschman gives in [6] a comprehensive description of architectural patterns and Paulo Serif provides in [7] a method for integrating architectural patterns in applications and applying pattern oriented architecture.

For setting the base framework structure we identified the following architectural patterns as useful and integrated them into IMM:

- Builder – used for instantiating messaging components depending on the current configuration settings.
- Proxy – used for implementing the proxy of the SIP server
- Observer – used for implementing all the notifications in the system
- Strategy – used in the client application for decoupling the client components from the media services.
- Broker – used for decoupling multimedia clients from the multimedia server and also for dynamically adding new multimedia clients.
- Reflector and Configurator – used for detecting, linking and unlinking at runtime the components implementation.

To integrate the architectural patterns into IMM we have followed the structured approach proposed in [7]. The pattern integration in the overall structure can be seen as a pattern itself having the following structure:

![Architectural Pattern Diagram](image-url)

Fig. 6. Integrating architectural patterns
To describe the integration of an architectural pattern we will use:

- An informal description that contains the usage context of the pattern, the intent, the problem, the acting forces and the consequences of using the pattern.
- Interfaces describing the interaction of the pattern with other architectural components. From the architecture point of view we have interface classes and interface operations. Interface classes are responsible with integrating the pattern into the context. Interface operations are implemented by the internal classes.
- The pattern structure describes the internal design components. In OO terms the internal structure is represented by class and collaboration diagrams.

As an example of integrating architectural patterns into IMM we have chosen the Broker pattern. The use of Broker has many justifications emerging from the following requirements:

- The clients connect to the multimedia server from different locations, with the server location transparent to the clients.
- Clients from different software providers implementing SIP should be able to connect.
- Scalability in terms of allowing several multimedia servers and load balancing.
- Support of clients having different multimedia capabilities.
- Support of H.323 clients and transfer from signalling protocols like H.323 to SIP
- Support for dynamic adding of new multimedia codecs and exposing them as services to clients

The next paragraph presents the integration of the Broker pattern into the IMM architecture.
4.1 Integrating the Broker Pattern

4.1.1 Broker Overview
In [6], Buschman states that the Broker architectural pattern is intended to be used to structure distributed software systems with decoupled components that interact by remote service invocations. A broker component is responsible for coordinating communication, such as forwarding requests, as well as for transmitting results and exceptions. The succinct, informal description and the structure of the Broker pattern are presented next:

**Problem and Forces**

The Broker is used in the context of distributed and heterogeneous systems containing interacting components.

The Broker pattern balances the following forces:

- Components should be able to remotely access services provided by other components, no matter their physical location.
- Components should be added, deleted, changed at run-time.
- The implementation details of the components should be hidden by the architecture to the component users.

**Solution**

- The Broker component decouples clients and servers.
- Servers register themselves to Brokers and provide their services through interface methods.
- Clients access servers’ functionalities sending requests to the Broker.

To realize the Broker pattern, six object types are involved: Client, Server, Broker, Bridge, Client-side Proxy, and Server-side Proxy. The static structure of the pattern is presented in Fig. 7.

The objects have the following responsibilities:

- **Client** – objects, components or applications invoking services of at least one server. Client subscribes to a Broker and sends its requests to the Broker. The Broker will send back the results or exceptions from the server.
- **Server** – implements Objects that provides functionalities and attributes exposed via interfaces. The Server registers to a Broker to receive requests from clients.
- **Client-side Proxy** – decouples the Client and the Broker in order for the client to access the Broker’s functions location-independent invoking the functions of the local Proxy.

**Fig. 7. Broker Pattern structure review diagram**

- The Proxy provides the following functionalities:
  - Inter-process communication between Client and Broker
  - Communication resources allocation and deallocation
  - Inter-process parameters marshalling
- **Server-side Proxy** – decouples the Broker and the Server.
- **Bridge** – optional components used to isolate implementation of two interacting Broker components.
- **Broker** – is an intermediate component sending requests to servers and responses back to the clients. The Broker should also implement a localization mechanism of the message receivers. Brokers should also be able to interact. Other functionalities like name services or marshalling can also be assigned to Brokers.

4.1.2 Broker Interactions

The Broker intermediates the communication between clients and the Media Server. In case of SIP-enabled clients the communication is direct but in case of the other clients, messages will be translated using the bridge. In our specific case the external clients are H.323 clients and the bridge is the Translator Server SIP/H.323.

Based on these interactions we can infer the internal interfaces required to the broker and the external interfaces exposed by the components it collaborates with.

The detailed design of the interface classes for the Broker would be too large to include in this paper, therefore we’ll detail only the interface with the IMM Client(s) instances. The Interface classes and their responsibilities are straight forward, and in order to further establish the Interface operations we’ll detail the dynamics of a multimedia call from an IMM Client to a Media Server.
The Client Application, located on the user machine, will interface with the Broker through the Client Proxy. At the implementation level the Client-side Proxy will implement the interfaces exposed by the Broker. Depending on the Client type and capabilities, implementation will vary, by example the Client::StartTask() will differ from video messaging to text messaging as well as the associated Client-side Proxy::PackData().

When analyzing the dynamic behaviour we need to discuss two collaboration scenarios: one regarding the Client Authentication and the other regarding the actual sending of the MM message. These two scenarios can be handled either by a single or by separate Brokers. In our implementation we have preferred, for Client-side Proxy simplicity, to have both handled by the entry point Broker which will forward, using Bridges, the messages to the appropriate SIP Multimedia Messaging Broker (IMM Broker), H323 Broker or Authentication Broker.

In the Authentication scenario the Client starts an Authentication Request and the Proxy forwards it to the IMM Broker, which sends it forward through the Bridge to the Authentication Broker. The Authentication broker validates the user credentials and if it succeeds returns a user session ID and, if message encryption is required, the certificates required for the secure communication. If validation fails, the Client’s access to the Broker is denied. The IMM Broker has a lookup loop which runs validation on logged users, for the scenario when a user’s access is denied while he or she are logged in. If the validation fails, the user’s subscription to the IMM Broker is marked as invalid and the IMM Broker replies to the received requests with an <account disabled> message.

In the Multimedia Messaging Scenario when a <send> message request is issued by the Client, this forwards it to the Client-side Proxy.

5 Conclusion

Software reuse is one of the most appealing yet difficult to achieve promises of Object Orientation. Building reusable pieces of software and software based on reusable components should be a systematic activity that affects all the system's levels of abstraction - from knowledge reuse in analysis to code reuse in implementation.

In [8] Christopher Alexander says that "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice". Even if he was referring to patterns in civil engineering, the principle stays valid in software development with the comment that the patterns’ interpretation and the actual implementation of patterns in software depends upon the level of abstraction on which they are used. In this paper we have studied architectural patterns, components with a high level of abstraction that help solving problems of structuring, communication, adaptability or distribution of complex software systems.
This article describes an example of a pattern based, reuse oriented architecture, applied for the development of an integrated multimedia messaging (IMM) system, that was developed under a joint effort between the Technical University of Cluj-Napoca and iQuest Technologies in an effort to increase the common expertise in both software reuse techniques and in the new areas of Internet service integration.

We have chosen SIP as the signalling protocol for our system because the advantages it offers for the application development:

- Complete split between the transport and application services which are regarded as different businesses running on top of different technologies.
- Service promiscuity: anyone can access services brought by any providers, anyone with IP connectivity can become a provider.
- Setting up a signalling service as easy setting up a web server.
- The service market is completely open.

Based on the SIP advantages our system offers integrated multimedia messaging and through its pattern based, reusable architecture, a structure for developing application families for integrated IP based services that can include, but are not limited to, instant messaging, video streaming, unified messaging (voice2email), directory services or calendars.

Because SIP offers the possibility to leverage its infrastructure for any distributed applications based on the notion of session and because we envisioned an application that will easily accommodate future services, we have developed our system using a systematic reuse driven approach. This involved an implementation of a reuse driven software development methodology and a reuse driven architecture development as described by Jacobson in [10] combined with the pattern oriented software architecture principles described by Buschman in [6].

The architecture development was started by investigating several architectural patterns:

- Structural patterns (Layers, Pipes&Filters, Blackboard).
- Distributed patterns (Broker).
- Interactive systems (MVC, PAC).
- Adaptable systems (Microkernel, Reflection).

and decided upon a pattern language applicable to our context.

We have detailed the layered architecture for both the overall system and for the Client Application subsystem and applied the method for composing patterns in software systems described by Serif in [7].

The architecture presented here showed the overall system structure and presented the elements of integrating architectural patterns into it as an architectural pattern itself. We then applied this integration pattern by providing an example of the integration of the Broker architectural pattern into the IMM.

References:

[3] Paulo Alencar, Donald Cowan, Jing Dong, A Pattern-Based Approach to Structural Design Composition, Department of Computer Science University of Waterloo, Carlos Lucena Departamento de Informatica PUC-RJ Rio de Janeiro.