The ITB Journal

Volume 5 | Issue 1 Article 29

2004

Developing Real-Time Multimedia Conferencing Services Using Java and SIP

Gavin Byrne
Institute of Technology Blanchardstown, Ireland., gavin.byme@itb.ie

Declan Barber
Institute of Technology Blanchardstown, Ireland., declan.barber@itb.ie

Developing Real-Time Multimedia Conferencing Services Using Java and SIP

Gavin Byrne and Declan Barber
Institute of Technology Blanchardstown, Ireland
gavin.byrne@itb.ie declan.barber@itb.ie

Abstract

This paper examines Java's suitability in creating real-time multimedia communications-based applications in Next Generation Networks (NGNs). We investigate some of the current enabling technologies provided by the Java platform which are concerned with the rapid development of real-time communications-based products and services. In particular, we look at creating a multiparty conferencing and collaboration service using the Session Initiation Protocol (SIP) and the JAIN Framework and present an approach which models multiparty conferencing applications by separating signaling and media transfer functionality. We map our model through the design stage to an implementation in Java. This paper is based on real experiences derived from work on an applied research project which is concerned with the development of a collaborative system which allows multiple distributed scientists to graphically analyse a common set of data obtained from mobile sensors in a virtual conference environment. Potential applications areas include education, emergency response services, gaming and any general collaborative application

Introduction

The Internet is steadily becoming more capable of providing real-time media distribution (voice or video) between participants. This functionality represents a significant enhancement to more traditional asynchronous conferencing functionality typified in message boards and chat rooms. One reason for this improvement is the increased bandwidth availability arising from broadband access and the promise of 3G mobile systems such as the Universal Mobile Telecommunications Systems (UMTS). This increase in bandwidth will continue to be a key factor in the increased use of media rich real-time conferencing. Another reason is the increasing support for real-time media transfer and signaling made possible by the trend towards convergence in previously diverse networks. Another third reason is the increasing support for real-time media transfer and signaling provided by open and standard Internet Protocols. Although Java has already been widely adopted for developing high-level business applications which leverage widespread Internet protocols such as HTTP, the complex and proprietary nature of underlying network technologies has meant that the creation of more flexible and granular communications services is still largely the domain of service provider or network equipment vendor personnel using highly specialised languages. This is changing with the emergence of new internet protocols and the Java Intelligent Networks Framework (JAIN), which increasingly allows third party developers to develop and deploy lower level communications services using high-level programming techniques.

This article suggests that Java is an excellent choice for developing end-to-end applications and services for the NGN (Next Generation Network) and will enable third party developers to offer new and innovative services independently of service providers. We describe our

approach to the creation of one such service using Java and the JAIN Framework and make observations on our experiences that may be more generally applicable to a range of other created services. This paper makes certain assumption:

- Bandwidth availability will steadily increase from end-to-end and decreasingly represent a technical constraint
- Convergence in the NGN will be based firmly on open standards and the TCP/IP protocol stack in particular
- Applications leveraging existing and emerging Internet Protocols will dominate

The JAIN Framework

The objective of the Java Intelligent Networks Framework (JAIN) [3, 5] is to provide service portability, convergence and secure access to integrated networks. JAIN builds on Java portability by standardizing the signaling layer of the communications networks into Java language and defining a framework in which services can be created, tested, and deployed. The JAIN initiative brings new opportunities for both developers and service providers, enabling them to create services (without rewriting) for the different network implementations and interfaces in a multi-vendor environment. The JAIN initiative proposes to do this by specifying Application Programming interfaces (APIs) which provide access to functional objects such as Call Control or User Location Objects as well as underlying signaling objects (e.g. SIP, H.323 and MGCP). This essentially allows application developers easy access to functional and signaling interfaces. In the context of our multiparty conferencing service, the direct access to signaling using SIP was a critical feature in service development.

Fundamental Design Issues

Our analysis of design principles and functional requirements is summarized in the following description of our conceptual model. Our conferencing system had the following general design aims:

- ***Scalability*** in the number of (distributed) users
- ***Efficiency*** in its use of network and node resources
- ***Simplicity*** in the entire service creation process and especially implementation
- ***Extensibility*** for the easy and rapid deployment of modifications or new services
- ***Interoperability*** of applications and underlying services based on NGN models.

Specific conferencing functional requirements include:

- Peer-to-peer and multiparty conferencing in a distributed NGN environment.
- Dynamic and flexible conference signaling and media transfer

- Real-time Voice/Video Support: it must allow effective real-time conferencing.
- Application platform independence

The adoption of open and standard internet protocols for real-time media transfer and signaling (specifically the Real-Time Protocol, RTP, the Real-Time Control Protocol, RTCP, and the Session Initiation Protocol, SIP) combined with the convergence of public and private networking technologies towards the internet protocol stack in general ensures that most of our design principles are achieved. From an application developer's perspective, these protocols are highly accessible through Java and its associated Application Programming interfaces (APIs). The two main functions within a conferencing system are Signaling (for call establishment, the addition/deletion of call parties and call termination) and Media Transfer. For simplicity and efficiency, our design further separates the signaling functionality into User Registration and Call functions. Consequently, our preferred approach uses a hybrid signaling architecture (Figure 1): a centralized signaling architecture supports user registration and directory support while a distributed signaling architecture allows peer-to-peer and peer-to-multi-peer calls. A distributed architecture supports peer-to-multipeer media delivery. This is scalable for large numbers of participants and makes efficient use of bandwidth and processing.

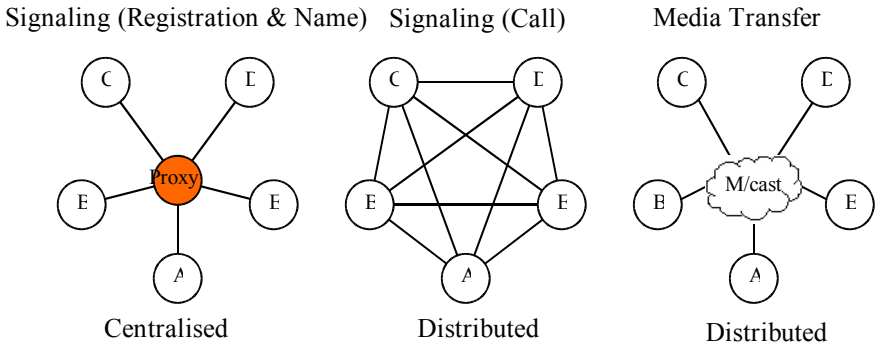

Figure 1: Conferencing Model based on Unicast Signaling and Multicast Media

Evaluating Java Technologies for Multiparty Conferencing

Java strongly supports the development of new applications and services which leverage the functionality of the new powerful Internet signaling and media transmission protocols. Java's platform independence, downloadability, mobility and object oriented structure has already led to its adoption for use in telecommunication applications and is destined to play a critical role in the development of internet-based electronic commerce systems. Java provides support for a wide variety of internet protocols such as HTTP (within applet/servlet packages), SIP (within JAIN Framework), RTP (within JMF package), IP (within net package), which allow development of inter-networked applications. The Java API's of most importance in creating multimedia conferencing applications are:

The Java Network package (java.net.*): Through the java.net package, Java provides the ability to create both unicast and multicast sockets for the transmission and receipt of data. The ability to create multicast sockets will be an advantage in our conferencing and collaboration application where identical data is being sent to multiple recipients as multicast is far more bandwidth and processor efficient (and therefore scalable) than having to open up multiple unicast sockets for the same data.

The Java Media Framework (including javax.media.*, javax.sound.*): The Java Media Framework (JMF) is a set of Java APIs for developing multimedia applications. In this project JMF provides the necessary methods for the transmission and receipt of real-time media streams using Real Time Protocol (RTP) and the Real Time Control Protocol (RTCP).

The Java Intelligent Network Framework (including javax.sip.*, javax.sdp.*): The Java Intelligent Network Framework (JAIN) includes a set of Java technology based APIs which enable the rapid development of Next Generation communications-based applications and services on the Java platform. By providing a new level of abstraction and associated Java interfaces for service creation across point-to-point, circuit-switched (PSTN, ISDN), and packet/cell-switched (X.25, Frame Relay, ATM) networks. Importantly for us, JAIN provided the only practical means of accessing the signaling using a high-level language and of separating the registration aspect of the signaling from the call-establishment.

Session Initiation Protocol (SIP)

SIP [9] is an application layer signalling protocol which provides call set-up, modification, and termination, as well as other services. Importantly, participants can communicate using multicast, unicast or a combination of both. As an application layer signaling protocol used in a distributed architecture, SIP is best suited to meet our scalability, real-time, simplicity and extensibility design requirements. The Session Description Protocol (SDP) is used in conjunction with SIP for exchanging session capabilities (ability to send/receive audio or video, supported codecs, etc.).

We believe that SIP will be the protocol of choice for Next generation Networks and we have chosen SIP to develop our multimedia conferencing application because of its easy integration with existing IETF protocols, simplicity, mobility, scalability, ease of development, extensibility and deployment in the core and at the edge of the enterprise and support for multicast, unicast or a combination of both (all documented and discussed in [4, 8, 9, 10]).

From Analysis to Design

In order to map our Unicast/Multicast model to a design and implementation we have used Sties and Keller's Independent Service Description Model [11] which consists of abstract descriptions for endpoints, communication channels, and communication relations. This is an abstract service model that allows the standardized description of the specific characteristics of a service while abstracting from any network and implementation dependant details.

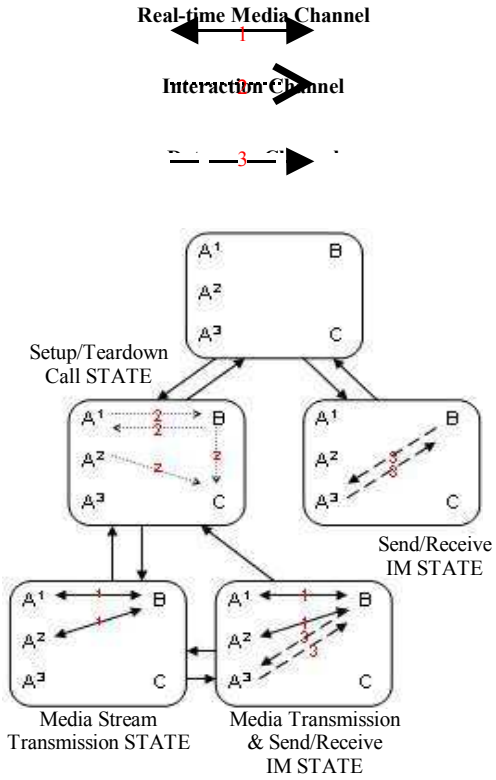

Figure 2: Finite State Machine

This approach begins by determining the functions of the service from the user's point of view: peer-to-peer call, conference call or instant messaging (labeled as A^1, A^2 and A^3 respectively in Figure 2), and then determining the endpoints needed within the service: a conference server (labeled C) and another human participant (labeled B). The next step is the identification of communications channel types. There are three identified in our service, as shown below. The final step is to identify all Single Communication Relations, i.e. Endpoint - Communication Channel relations that might occur in our service (identified as the states in Figure 2). We then summarise the Single Communication Relation States found into an FSM (Finite State Machine that summarises our system).

We now employ the State pattern described in [13] which can be used to translate an FSM into a set of high level abstract classes and relationships. We begin with a class named UserAgent,

which creates and manages 'manager' classes for each of the areas of functionality which we can now identify using our FSM (call setup/teardown, media transfer, and Instant Message exchange). The UserAgent class passes events and method calls to the appropriate manager to deal with, then simply updates a global state variable and awaits further events. Figure 3 illustrates these classes in a UML diagram.

Implementation

From our analysis and design, we have identified what states our SIP user agent can be in at any one time, as well as what classes and objects need to be implemented. Our SIP user agent is implemented as a pure Java applet, which is digitally signed to enable the applet some permissions which are usually outside the Java security sandbox (such as access to hardware like microphones and web cams), allowing it to be used in thin web clients.

The key features implemented in our user agent thus far are the ability to **A)** Register with a SIP proxy/registrar server, **B)** to make voice and video calls, **C)** to send and receive Instant messages (Interoperable with MSN Messenger, etc.), **D)** to add contacts to a buddy list with presence capabilities (which informs the user when contacts are on or offline), **E)** to make conference calls using unicast call signaling to the conference server and multicast media transmission, and **F)** to collaboratively analyse remote sensor data stored in a web server database through graphs and charts with other users in either conference calls or peer-to-peer calls allowing users to highlight (by drawing on the graph) interesting findings which is replicated to all interested parties.

Through the JAIN SIP 1.0 and JAIN SDP APIs we have been able to harness the simplicity of SIP for call signaling, as well as buddy lists [6, 7] and Instant Messaging [1] (using the SIP MESSAGE and SUBSCRIBE extensions) in our conferencing system. The user agent and conference server are developed on top of the publicly available NIST (National Institute of Standards and Technology) pure java sip stack implementation [12].

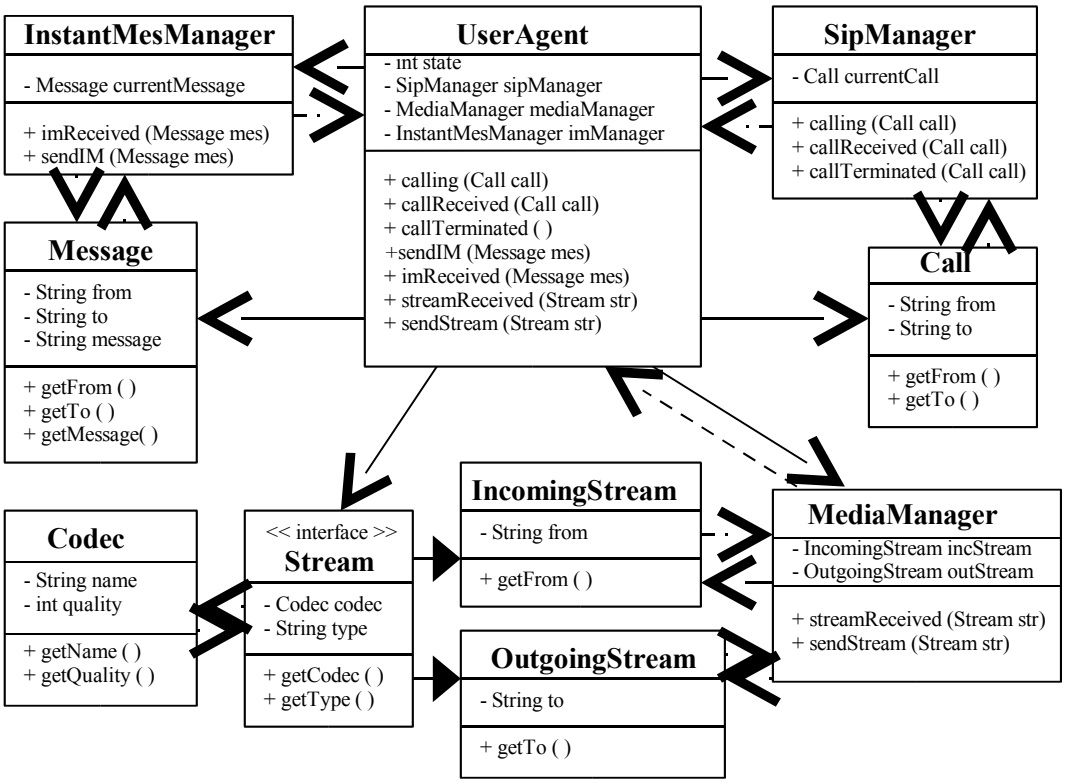

Figure 3: UML Class Diagram

Instead of developing our own SIP proxy server at this stage, we are using a proxy server which was freely available, again from NIST.

Figure 4: Implementation Screenshots

Users wishing to join the conference can simply send a standard SIP INVITE message to the conference server which in turn can choose to authenticate the user or simply send an immediate ACK reply to set-up the call. Users currently participating in a conference who would like to invite other users into the conference can send a SIP REFER message with the URI of the conference server, inviting them into the call (this REFER message could alternatively be sent to the conference server with the intended recipient's URI). Each

participant makes a normal peer-to-peer SIP call to the conference server using unicast signaling to register initially or to resolve a SIP name to an IP address. A unicast call is then made to the peer. Once the call is established, media is sent and received on a multicast connection. Other registered peers can be invited or proactively join the established multicast conference. The role of the conference server is to act as the centralized manager of the conference, and to maintain a signaling dialog with each participant in the conference

Summary of Observations on Using Java & JAIN for Conferencing services

Strengths and limitations of using Java technologies for conferencing service creation include:

- Java's ability to send and listen for data on multicast sockets was a major benefit.
- The JAIN framework provides a solid set of APIs for the implementation of new communications-based services and will increasingly enable developers to rapidly create and deploy new services without specialist knowledge.
- The real-time streaming abilities provided by the JMF are slow to initialize and begin streaming, but once started provide a perfectly adequate solution. It compares poorly to similar Java telephony applications which make use of native code for access to hardware, for example the DLL (Dynamic Link Library) for the windows platform. These manage almost instantaneous responses. The use of native code in our clients would contradict our platform independent model, but in order to provide realistic response times, we may have no choice.

Conclusions

- Java's platform independence, downloadability, mobility and object oriented structure has already led to its adoption for use in telecommunication applications and is destined to play a critical role in the development of internet-based electronic commerce systems. But it is Java's ability to leverage existing and emerging internet-based open protocols, and the fact that it is now enabling third party developers with little or no knowledge of underlying network infrastructures to develop and offer new services independently of service providers, that will no doubt make it the language of choice for developing applications and services for the NGN.
- The ability to replicate multimedia conferencing capabilities currently available in the circuit-switched environment will not be a sufficient driver for the enterprise to adopt IP based multimedia conferencing. The ability to rapidly develop and deploy enhanced, and previously unavailable, services which leverage IP and related intelligence will be the driving force behind the evolution of NGNs. Java and the JAIN framework with its power to implement emerging internet protocols such as SIP is a key catalyst in this evolution.

References

[1] B. Campbell et al., "SIP Extensions for Instant Messaging", IETF DRAFT, work in progress, 2002.

[2] Sun Microsystems, "JAIN Service Creation Environment (SCE) API Specification".

[3] Sun Microsystems, "Java Advanced Intelligent Network, The JAIN API's".

[4] I. Miladinovic, J. Stadler, "Multiparty Signalling using the Session Initiation Protocol"

[5] P. O'Doherty, M. Ranganathan. "JAIN SIP tutorial".

[6] J. Rosenberg, "The Future of SIP and Presence", 2003.

[7] J. Rosenberg et al., "SIP Extensions for Presence", IETF DRAFT, work in progress, 2001.

[8] J. Rosenberg, H. Schulzrinne, "Modles for Multi Party Conferencing in SIP", IETF DRAFT, work in progress, 2001.

[9] H. Schulzrinne et al., "SIP: Session Initiation Protocol", IETF DRAFT, November 2000.

[10] H. Schulzrinne et al., "Centralized Conferencing using SIP", IETF DRAFT, November 2000.

[11] P. Sites, W. Keller, "A Generic and Implementation Independent Service Description Model".

[12] National Institute for Standards and Technology, "SIP Reference Implementation", http://snad.ncsl.nist.gov/proj/iptel/

[13] R. Martin, UML Tutorial: Finite State Machines, Engineering Notebook Column, 1998.