BUILDING A SCALABLE FRAMEWORK FOR THE COLLABORATIVE ANNOTATION OF REAL TIME DATA STREAMS

TAO HUANG

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
int the Department of Computer Science
Indiana University
January 2013

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Professor Geoffrey C. Fox, Chair

Professor Gregory J. E. Rawlins

Professor Minaxi Gupta

Professor Kay Connelly

January 11th, 2013

ACKNOWLEDGEMENTS

This dissertation would not have been accomplished without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this research. I would like to give my sincere thanks to all these remarkable people.

First and foremost, my utmost gratitude to my advisor Prof. Geoffrey C. Fox, distinguished scientist and director of Pervasive Technology Institute at Indiana University, for his encouragement and guidance in the completion of this dissertation and the exceptional research environment he offered. His perpetual energy and enthusiasm in research had motivated all his advisees, including me. In addition, he was always accessible and willing to help his students with their research. As a result, research life in Bloomington became smooth and rewarding for me.

I was delighted to attend Prof. Gregory J. E. Rawlins's course and have in-depth discussion on system design and implementation with him. He sets an example of a respectable researcher for his rigor and passion on research.

Prof. Minaxi Gupta and Prof. Kay Connelly deserve special thanks as my thesis committee members and advisors. I could not finish this dissertation without them generously offering time, support, guidance and good will throughout the preparation and review. I am thankful for their suggestions, constructive comments, kindnesses and keen intellects.

I would also like to thank all members of Community Grids Lab for the priceless moments that we shared together. It has been a great honor to work with these extraordinary people. I particularly thank Prof. Shrideep Pallickara from Colorado State University for his reviews and productive discussions on extensive areas of my

research. I thank Ms Mary Nell Shiflet for her prompt help on my scheduling and coordinating requests on research resources.

Last but not least, my gratitude to my family for their unflagging love and support throughout my life, this dissertation is simply impossible without them. My parents, Peizong Huang and Qingli Peng, have been a constant source of emotional, moral and of course financial support during my postgraduate years, I could hardly image how far I could achieve without them. My wife Juntao Yu has been, always, my pillar, my joy and my guiding light, I would not have finished this degree without her encouragement.

TAO HUANG

BUILDING A SCALABLE FRAMEWORK FOR THE COLLABORATIVE ANNOTATION OF REAL TIME DATA STREAMS

With the fast development of Internet technology, competent on-line collaboration tools are currently being used daily to improve group productivity. There are no longer such limitations for people to work on digital contents using designated platforms on specific networks. They can choose different collaboration services with diverse connectivity and user interface options. As an important feature of on-line collaboration systems, collaborative annotation on real time streams is being thoughtfully investigated and studied by different research entities all over the world. Systems such as Vannotea[*Schroeter et al.*, 2003] and SIDGrid[*Bertenthal et al.*, 2007] have been invented for common collaboration requirements such as audiovisual communication and digital annotation, and they have accomplished the targets very well. However the majority of such systems are designed and implemented to process merely specific data such as multimedia streams, which can be hardly extended to support generic contents such as real time data from earthquake sensors, traffic monitors and medical instruments. It is challenging to design and develop such a framework that supports creating, sharing and replaying annotations on generic data streams regardless of end user's multiplicity of connectivity and supporting platforms. In this dissertation, we investigated major characteristics of popular collaboration and annotation systems on both desktop and mobile platforms, summarized key requirements and research difficulties of building a distributed collaborative annotation framework, and then presented our prototype of such a system that supports annotating generic data streams in heterogeneous environments. The analysis of experiment results demonstrates that our decisions on the system architecture and design have provided various advantages over previous systems on both performance and scalability.

TABLE OF CONTENTS

CHAPTER

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER I

Introduction

1.1 Annotation in Distributed Collaboration

Internet has evolved tremendously during the past decade, it becomes the most important platform for information publication, sharing and servicing. With the deployment of high speed networks such as Internet2[*Kratz et al.*, 2001] and 4G LTE[*3GPP*, 2010], it becomes possible to access and process large amount of digital data despite location and time constraints. People tend to move their personal data and computational jobs into Cloud platforms such as Amazon S3[*Amazon LLC*, 2006], Google Drive[*Google Inc.*, 2012] and Microsoft Azure[*Microsoft*, 2010], therefore they can utilize the best computational and storage resources to serve their purposes with minimum cost. In addition to storage and computation, on-line collaboration is also an importation service necessary to Internet users. It helps resolve geographic, time and communication difficulties that people may encounter during their inter-organizational cooperation.

During the past decade, various distributed collaboration platforms[*Childers et al.*, 2000][*Schroeter et al.*, 2003][*Bertenthal et al.*, 2007] have been designed and implemented to help people in accessing, editing and collaborating on data of their interests easily. Among all popular cooperation activities, annotation is one of the most commonly conducted tasks and therefore becomes an obligated feature of many collaboration systems. It is also important for these platforms to support easy integration of new types of content data and have user friendly desktop and mobile interfaces.

Annotation in general is defined as the process of adding opinions, comments, making notes or explanations to portions of content data. It is being used in different application areas, varying from interpreting plain texts to commenting on multimedia clips. People can choose to annotate their content data in multiple dimensions such as textual criticism in 1-Dimension annotation on literatures and cartography in spatial annotation. From most annotating activities being conducted over the Internet today, we have made a conclusion that most of them focus on static data which are prerecorded and preprocessed with relevant meta-data for analysis, processing and reviewing afterwards. This simple "Record-Annotate-Replay" routine is useful and effective for academic and research purposes such as training, weather forecasting and genome decoding[*Stein*, 2001]. However it may not be suitable for scenarios that require annotating and analysis on data streams being generated in real time.

One good example of annotation on real-time/live data streams would be earthquake prediction. Thousands of earthquake sensors have been deployed in areas with frequent crustal motions, for example California, United States and islands of Japan. If a timely analysis and annotation on abnormal crustal activities could be done and presented to the authority, damages and casualties caused by disasters such as Tohoku earthquake in 2011 might be controlled at a minimal level. Another important case is traffic control in large cities. It is common that traffic control departments of major cities in the world are overwhelmed by live video feeds captured from all areas within the city. An automatic traffic monitoring and accidents reporting system can help them improve efficiency and accuracy enormously. Most recently, with the widespread use of smart phones and hand-held devices, Near Field Communication[*Want*, 2011] makes eCommerce much simpler and faster. Monitoring live transactions data over the mobile network becomes important and useful to protect customers from identity theft and phishing.

Collaborative annotation defines joint commenting on the shared content data

by a group of users that have similar interests. It is important for the supporting platform to record both user inputs as well as the context data generated during the collaboration. Both information have the same the importance for the user to interpret the content. Taking on-line course training as an example: students may have questions on particular sections of the instructor's handouts or e-documents. Currently they can make comments or ask questions through web forums and emails, instructors can respond accordingly but the context of the question is lost after the conversation finishes privately. Other students cannot benefit from the discussion without digging into previous conversations and gaining the context. Even though there are limited implementation of such context preservation in some systems, they are still text oriented and cannot apply to generic data.

1.2 Motivation

There are three major motivations that drive this extensive research of a collaborative annotation platform on generic content data. Firstly there are high demands of collaborative annotation in various application areas, varying from production industry to academic research institutes. Large amount of streaming data are being generated and accumulated daily. It is impossible for a single person or group to store or process such massive data by itself. According to Youtube statistics[*YouTube LLC*, 2012], 72 hours of video are uploaded to its website every minute all over the world. 2.5 quintillion bytes of data such as climate information from sensors, posts to social media sites, digital pictures and videos, purchase transaction records, and cell phone GPS signals are created every day[*IBM Corp.*, 2012]. Mining and analyzing such big data is impossible for a single computing entity or person to accomplish without collaborative work. There are also cases, taking the Tohoku earthquake described in previous section as example, that joint efforts are required by multiple administrative authorities to handle disastrous emergencies with computer based aids.

Secondly, it would be much convenient if we are able to annotate on live data streams. Given those typical use scenarios presented in previous section, collaborative annotation on real time content will benefit lots of people with faster responses to their problems in reality.

Thirdly, current systems are developed separately to serve their own purposes on specific content data. Many duplicated efforts have been spent on similar functionalities among various systems. People will benefit a great deal after we summarize their similarity and design a universal platform with common collaboration features. A simple integration interface will increase the adaptability of the platform even more since now we can add supports to new types of content data without extra work.

Lastly, it is important to add support to mobile devices such as tablets and smart phones, giving the face that they are becoming the major accessing ends to live data streams such as video feeds and sensor data. [*YouTube LLC*, 2012] has stated that traffic from mobile devices has tripled in 2011 and more than 20% of global views are from mobile devices.

1.3 Research emphases

To address motivations listed in previous section, we identified following research topics varying from distributed data storage to universal data support.

1.3.1 Annotation on Generic Realtime Data

There are two ways of presenting annotations on content data, either embedding them within the actual content (for example annotations in Youtube, Wikipedia, Google Maps and Genome Annotations) or displaying them alongside the content (for example MRAS[*Bargeron et al.*, 2001], Vannotea[*Schroeter et al.*, 2003] and ELAN[*Berez*, 2007]). The following picture shows a side by side comparison of these two choices.

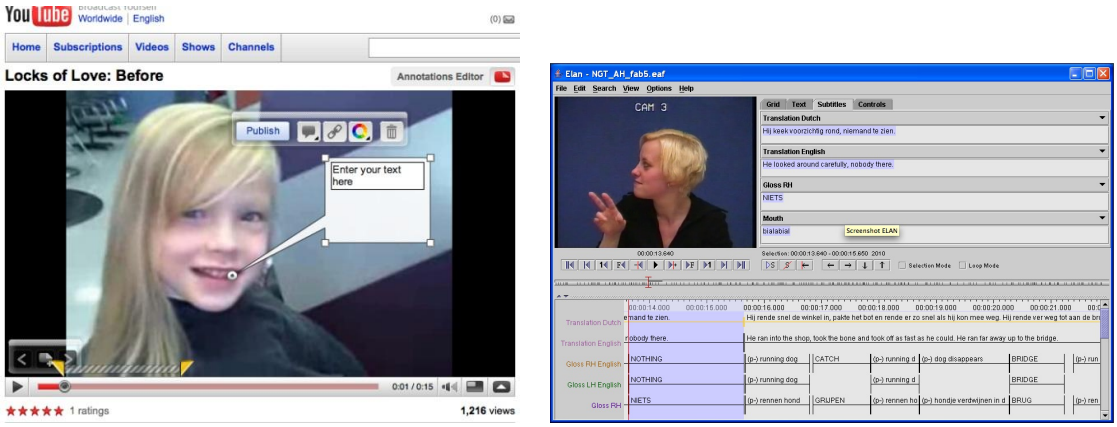

Figure 1.1: Two ways of presenting annotations

The first method helps users interpret the annotations at exact places, thus providing better semantic understandings. It is really helpful in multidimensional annotations such as cartography, but there are obvious shortcomings. For example, annotations may become too large therefore block the important portions of underlying contents. This sometimes may break the semantic connections between different parts of the content. The second method, on the contrary, keeps the integrity of the content and provides the semantic related information such as location through descriptive text comments. It is not as explicit as the first method and those comments may be ambiguous sometimes. Since the proposed platform is designed to support generic content data, it would be better to support both methods at the same time when necessary.

Unlike static content data, the status of real time data streams is transient and changes quickly. Some annotations are only meaningful at particular time spots during rendering of those streams, therefore it is required that the platform supports pausing/resuming the content streams while users are annotating them or viewing annotations already added. The status of the streams being annotated should be retained during those procedures and resumed later on. To solve this problem, three modes for annotations are proposed here:

- *Digress Mode*: In this annotation mode, the streams being annotated are paused to allow the annotation to be inserted and positioned. The newly received content data will be buffered locally and continued to be used for rendering after the annotation process is finished. This mode is generally used by annotating on prerecorded data streams and not recommended for real time streams since the local buffering will cause delay of rendering and therefore asynchronous status among different clients, which may break normal communication and correct understanding.

- *Commentary Mode*: In this mode, the rendering of content data streams will not be interrupted while being annotated. All annotations added will be received immediately and presented on the client immediately. Annotations that are unable to be presented in time will be stored and can be replayed when the users choose to switch to Digress mode.

- *Compound Mode*: This mode is a combination of above two modes which allows users to annotate on all streams in the session, including annotation data created by other users.

All these three modes will be applied to different scenarios of annotation and should be able to switch to each other according to the users choice.

1.3.2 Annotation Distribution and Storage

In order to ensure the fluency of annotating on the client side, both content and annotation data events will be stored locally in buffered files. Since they are also stored permanently on the remote storage service nodes, consistency must be kept so that every client node gets the same views of both content and annotations. There are two general methods of keeping such consistency, either through broadcasting updates from clients or mediating all nodes by the storage service. In the second

method, changes submitted from clients will be combined into one large batch event and pushed to clients periodically. Figure 1.2 depicts an example of both methods in a live annotation session.

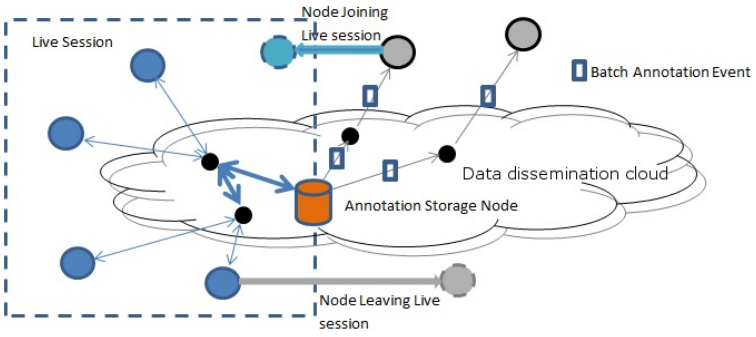

Figure 1.2: Methods of annotation distribution

Both methods will be provided in the platform for different scenarios such as live annotation and afterwards commenting. When users are collaborating in a live annotation session, it is important that they can see each other's modifications on the content data in time. Annotation events will be transmitted among them immediately without processing. Storage service node does make copies of these events and generate a long batch historical file for other users to view after the live session. Modifications occurred during the reviewing processes will be added to this batch event and then pushed to other session reviewers clients periodically. Session attendees will not know the content under viewing is modified until they restart the process or makes changes themselves. Through this method, the storage service can serve users in live sessions with more resources and attention.

Due to its simplicity and generality, most annotation systems define their own annotating languages based on XML. For example, MPEG-7[*ISO/IEC Moving Picture Experts Group*, 2004] standard from ISO/IEC is widely used among audiovisual annotation systems [*Savakis et al.*, 2003][*Ren and Singh*, 2005][*Bargeron et al.*, 2001] for fast and efficient searching for material that is of interest to the user. A set of Description Schemes and Descriptors are used to define descriptions of the underline

multimedia content data and schemes for coding those descriptions are also well standardized. Although the meta-data are stored separately from the audiovisual content, relations between them based on timestamps are created and stored inside the XML. Since we focus on annotating real time data streams, a similar scheme can be used and extended based on more generic RDF[*W3C RDF Group*, 2004] specification to meet our requirements.

1.3.3 Annotating Realtime Data in the Mobile Environment

Another interesting and important research is to extend the platform into mobile environments. Due to its limited computing resource and presenting layout, mobile devices can hardly provide the same level of user experience on annotating real time data streams as those applications on desktops/laptops. The instability of mobile networks such as 3G/EDGE also requires the platform to be more tolerant to high possibility of network losses and delays. Therefore several features with rich user experience described in section 1.3.1 should be translated to simpler presentation forms that are available on mobile devices, for example translating time-line based annotations to simple tags over the video clips. Data communications between annotation storage service and mobile clients might also need to be combined together at a higher level to minimize side effects caused by unstable mobile networks.

1.3.4 Platform Design and Implementation

The final objective of this dissertation is to develop a platform prototype which implements the common annotation capabilities on generic data streams in distributed environment. This implementation will serve as a workbench to test the design in terms of usability from both annotator and viewers points of view. The performance and scalability of the platform will also be tested with a number of clients working together within the Pervasive Technology Institute as well as outside campus of Indiana

University. Figure 1.3 below depicts an outline of the proposed framework.

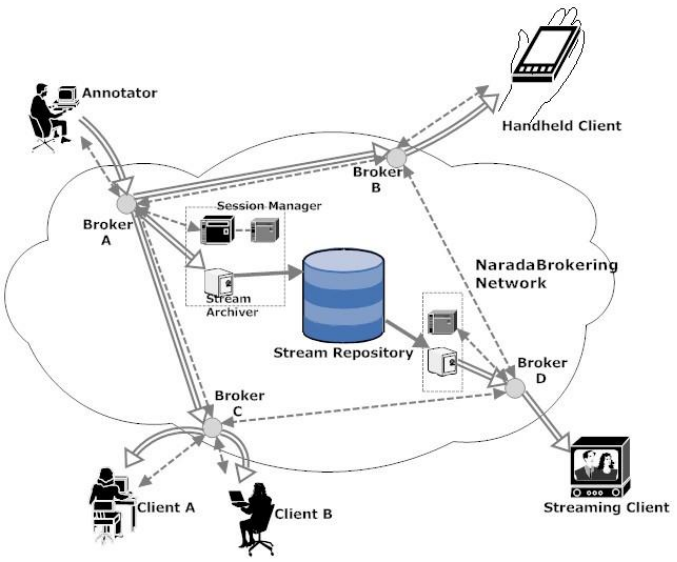

Figure 1.3: Platform architecture

As shown above, the platform is divided into several key components. A session management component is deployed on a dedicated server to control session events being generated from attending clients and service nodes. Stream archiving component can be deployed on different machines sending constant reports to the session server. Clients will be downloaded from the web portal with different plug-in based on processing requirements of different data types. And all controlling and streaming events will be transmitting over the data distribution network such as NaradaBrokering[*Pallickara and Fox*, 2003].

To support different types of streams in client, a web interface is presented to edu users for them to submit their own implementation of the streaming processing interface [*Huang et al.*, 2009]. They will be responsible for implementing stream processing plug-in of the platform and submit it through the web portal. Once the submission is done, the platform will be able to handle the requested type of real time data, and apply existing annotation functions on them.

1.4 Contributions

The major contribution of this research is to provide a scalable annotation framework of live data streams. It simplifies the efforts of collaborative annotation on real time streaming data, and it also presents an efficient system that supports cross-platform collaboration and a standard interface for generic data stream annotation. The profiling and evaluation being done in this research also show us an more accurate and efficient method for research entities to apply it on related collaboration systems in the future.

1.5 Thesis Organization

In this research, a comprehensive survey is firstly conducted on existing distributed annotation and collaboration systems. Through analysis of those systems, important features and requirements are summarized for building a scalable framework for the collaborative annotation of real time data streams. A prototype of such framework is designed and implemented to verify the correctness of our thoughts and methods in solving various research issues such as multi-dimensional annotation of real time data, flexible replaying of the annotation, robust annotation storage and distribution, simple but well defined interfaces for supporting new data types and so on. Finally, experiment results on the platform prototype testify the scalability of the framework, and some problems we encountered during annotating live data in mobile environments are also addressed.

CHAPTER II

Research Background and Survey of Related Technologies

Distributed collaboration and annotation systems have been developed in the past decade all around the world. These systems are designed to service different aspects of collaboration. Commercial H.323[*Karim*, 2000] systems such as Polycom and Tandberg provide most reliable audiovisual communication among heterogeneous networks and therefore dominate the video conferencing market. As a free alternative, Access Grid[*Childers et al.*, 2000] is very popular in the academic community. Scientific discussions and lectures are being held on this system almost every day. Besides video conferencing, document sharing and annotation is another major requirement of current collaborative annotation systems. Tools such as Google Docs[*Google Inc*, 2005] and Microsoft Office 365[*Microsoft Corporation*, 2011] become very popular for document based team work over the Internet. Gradually all these tools tend to absorb each other's popular features. For instance, Access Grid now has basic document sharing capabilities via its web portal while Google Docs users can video chat with each other through either the new Gmail feature or Google+ hangout.

2.1 Traditional Collaboration and Annotation Systems

Collaborative platforms that support annotation have already been developed by various software companies and academic groups all over the world. Several example systems are introduced and compared in this section to help summarize major features

that a collaborative annotation framework should have and problems that it should address.

2.1.1 H.323 and SIP systems

As the most commonly used video conferencing standard, H.323 is an International Telecommunications Union standard aiming at multimedia communication over packet switched Networks. It is defined as an umbrella standard which specifies its components to be used within an H.323-based environment. It provides conference management functionality for audio/video conferences using the call signaling functionality of H.225[*ITU Recommendation H.225*, 2000], H.245[*ITU Recommendation H.245*, 2000]. H.225 and H.245 provide call set-up and call transfer of real-time connections to support small-scale multipoint conferences. The H.243[*ITU Recommendation H.243*, 1998] protocol defines some commands between the H.323 Multipoint Control Unit (MCU) and H.323 terminals to implement audio mixing, video switch and cascading MCU. Codecs used within H.323 system are G.711[*ITU Recommendation G.711*, 1988] for audio, H.261[*ITU Recommendation H.261*, 1991] and H.263[*ITU Recommendation H.263*, 1998] for video and T.120[*ITU Recommendation T.120*, 1996] for data. T.120 recommendation contains a series of communication and application protocols. It also includes services to support real-time, multi-point data applications in collaborations sessions. Figure 2.1 below shows major components of a typical H.323 system.

As shown above, there are following six major components:

1. Terminals are endpoints that provide audio/video/data to another endpoint.

2. Gatekeepers (GK) provide admission and call control services to endpoint. They also provide services such as address translation, RAS control, call redirection and zone management to H.323 participants.

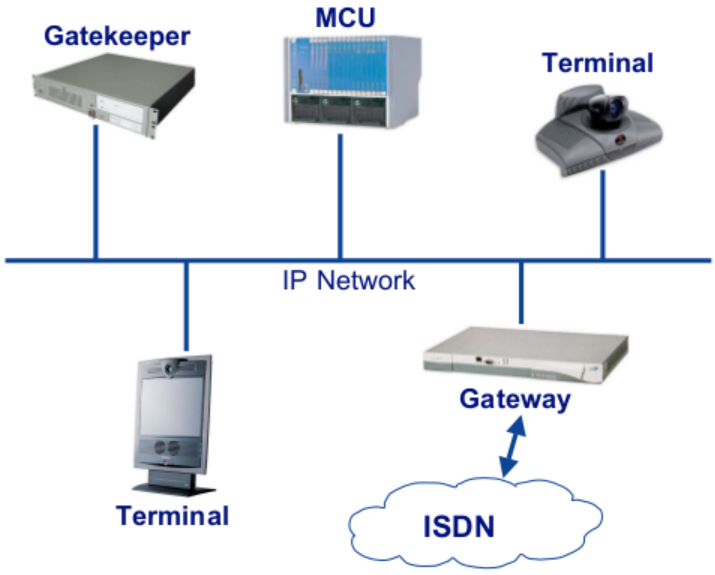

Figure 2.1: H.232 system architecture

3. Multipoint controller (MC) establishes a H.245 control channel with H.323 participant to negotiate media capabilities.

4. Multipoint processor (MP) provides media switching and mixing functionalities.

5. Multipoint control unit (MCU) is an endpoint that enables three or more endpoints to participate in a conference.

6. Gateway (GW) provides real-time, two-way communication between H.323 endpoints and non-H.323 endpoints.

H.323 protocol defines how these components communicate with each other, and the communication between them is defined in binary format. There are many popular products [*Polycom Inc*, 2000] [*IVCi LLC*, 2009] [*OpalVoip and H323Plus*, 2007] implemented based on this standard and they can inter-operate with each other.

The Session Initiation Protocol (SIP) is a text based and HTTP-like style (request-response) application layer protocol for establishing, modifying and terminating sessions. SIP was designed to solve problems for IP telephony. It provides functions

such as user location resolution, capability negotiation, and call management. SIP capabilities are basically equivalent to the services H.225 and H.245 in H.323 protocol. Although SIP does not define the conference control procedure like H.243, there are some researches for SIP based conference control protocol[*Koskelainen et al.*, 2002][*Wu et al.*, 2002].

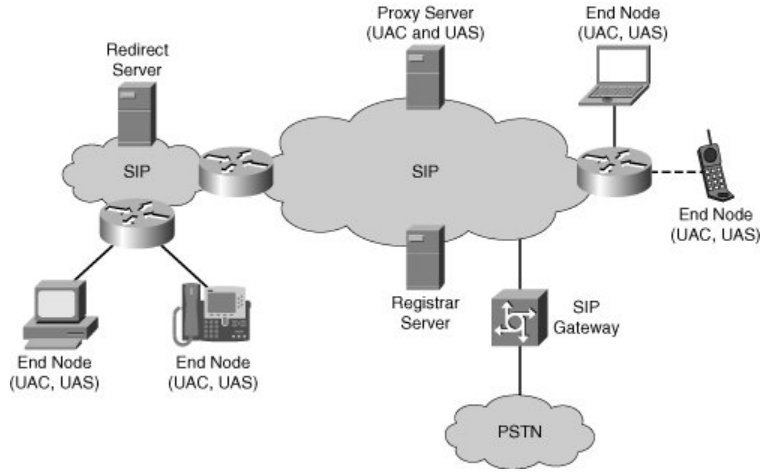

Figure 2.2: SIP system architecture

As shown in Figure 2.2, main components of a SIP system are: SIP clients, a SIP Proxy Server, a Registrar Server, a Location Server, a Redirect Server and a SIP MCU. The SIP Proxy Server primarily plays the role of routing and enforcing policy of call admission. It provides an instant messaging service, forwarding SIP Presence Event messages and SIP text messages to SIP clients. The SIP registrar accepts REGISTER requests and saves the received information in location server.

2.1.2 MRAS

Microsoft research released its annotation system MRAS[*Bargeron et al.*, 2001] in 2000, the system was designed to help Microsoft employees gain better training experience through asking questions on pre-recorded lecture videos. As show in Figure 2.3 The questions are anchored on the multimedia content and answered by the instructors asynchronously. Since the questions can be synchronously replayed with

the class content, students that have similar questions at the same time spot will benefit from reading answers to the previous question. Collaboration is achieved through discussions on the questions and their answers. MRAS doesn't support live video feeds and students who are watching the same video streams could not exchange their thoughts in the real time.

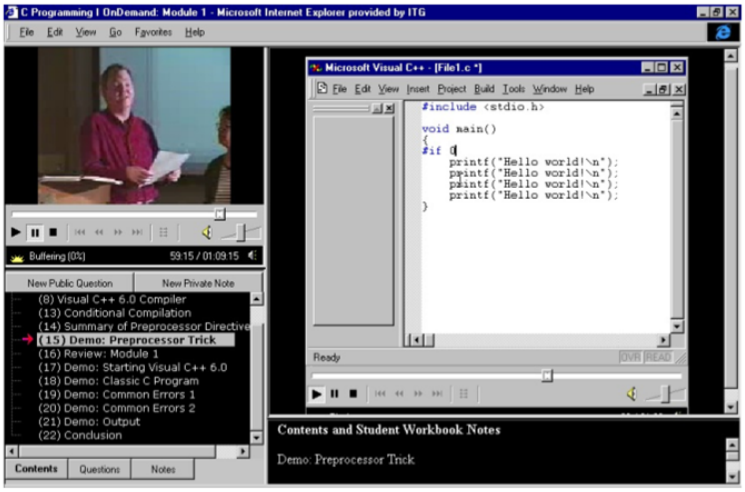

Figure 2.3: Microsoft Research Annotation System

2.1.3 VideoAnnEx

IBMs Mpeg-7 annotation tool VideoAnnEx[*Smith and Lugeon*, 2000] was also released in 2000. It can parse Mpeg video files and segment them into small shot units. Each shot unit can be annotated with a description from three default categories: static scene, key object and event. All shot units in Figure 2.4 are stored into a XML file as well as their descriptions/annotations following the Mpeg-7 standard. Users can search among the descriptions and replay the video shots alongside the description they are looking for. VideoAnnEx is a stand-alone annotation program that as well as MRAS cannot process live video feeds, and it does not support sharing and manipulating video streams among distributed users either. It can merely process MPEG-1 and MPEG-2 video files and the descriptions are limited to three pre-defined

categories. It is difficult to extend the system without modifying its source.

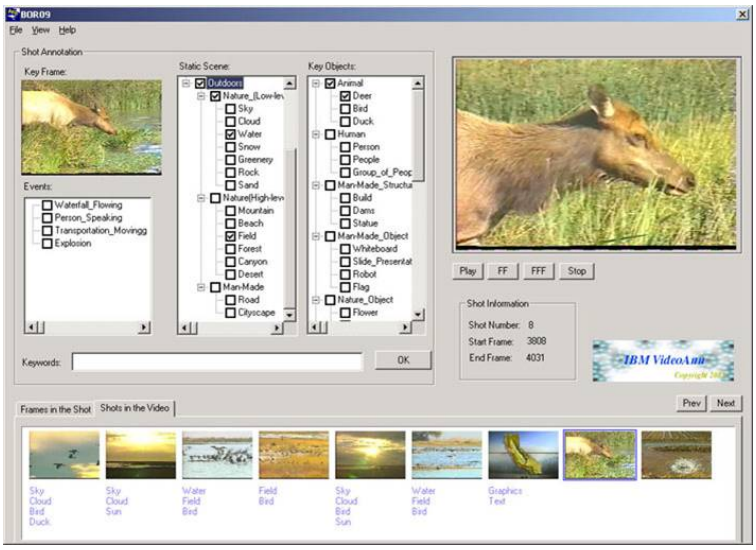

Figure 2.4: IBM VideoAnnEx

2.1.4 Vannotea

Researchers from University of Queensland invented Vannotea[*Schroeter et al.*, 2003] to help facilitate collaborative video indexing, annotation and discussion of video contents in a distributed broadband environment. It supports most features that VideoAnnEx has and provides more flexibility on the meta data of video segments. In Figure 2.5, Vannotea users are able to save, browse, retrieve and share both objective descriptions of the video files as well as subjective annotations on them. The videos files are still limited to Mpeg-2 format and users can only create text descriptions.

2.1.5 SIDGrid

The Social Informatics Data Grid (SIDGrid)[*Bertenthal et al.*, 2007] from University of Chicago is a new cyber infrastructure designed to transform the methods that are used by social and behavioral scientists to collect and annotate data, collaborate and share data, and analyze and mine large data repositories. It provides a

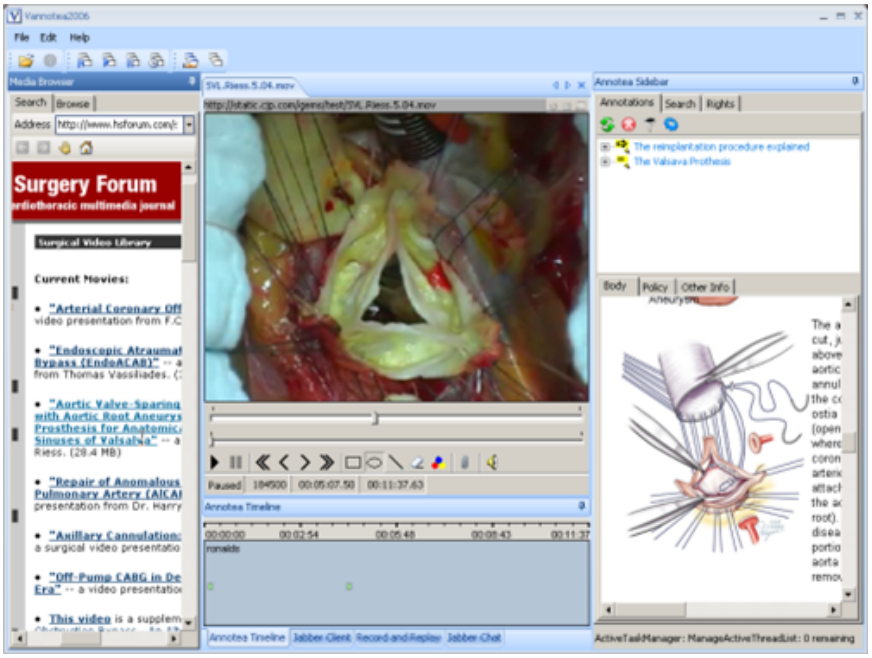

Figure 2.5: Vannotea from University of Queensland

novel integration of annotation, analysis, and search for multimodal data as well as a powerful framework for web-based, distributed collaborative annotation and analysis. As you can see from Figure 2.6, all annotation tasks are carried out through a modified version of the open source audiovisual annotation tool named ELAN[*Berez*, 2007]. Researchers can work with each other using a web based central archive of multimodal data, annotation and analysis. Though the browser-based interface helps achieve the collaboration objectives such as searching and discussion, it still cannot support annotating and analyzing data generated in the real time. And no collaborations on ad-hoc annotation are allowed, users can only work on prerecorded content.

2.1.6 A Collaborative Annotation Framework for Social Network Users

A collaborative annotation framework[*Shevade et al.*, 2005] from Arizona State University was proposed in 2005 to enable members of a social network to collaboratively annotate a shared media collection. It provides recommendations based on

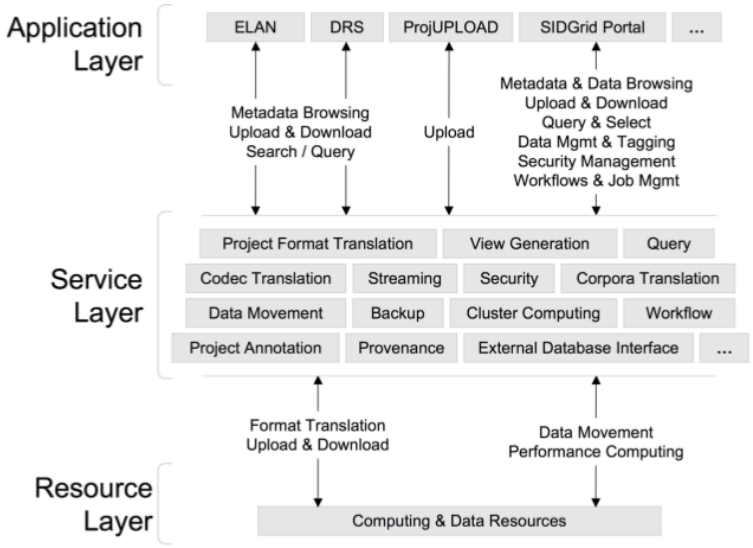

Figure 2.6: The Social Informatics Data (SID) Grid

low-level features, context, commonsensical and linguistic relationships. The framework firstly parses three major features of the shared media, computes the feature distances based on histograms of these features, and then uses a concept filtering method of the user context to adapt the final recommendation results. The research is mainly focused on its algorithm of generating more semantic relevant recommendation to avoid useless new annotations. The algorithm heavily depends on features of the shared media which are uploaded images and cannot be used on real time data streams.

2.1.7 eSports

eSports[*Zhai et al.*, 2005] developed by Community Grids Lab is another attempt to enable collaborative annotation on multimedia content over the distributed network, especially the grid-computing network. It enriched the annotation on multimedia contents from simple text to more diverse forms such as graphic shapes, audio/video clips. As its name indicates, eSports system aims to help sport coaches train their trainees remotely through vocal and graphic annotations on real time or archived video streams. As the Figure 2.1.7 shows below, coaches can take snapshots

of sample gestures in the video and comment on them to help students understand their classes. Annotations and video streams are archived using NaradaBrokering storage service and can be replayed synchronously based on their time stamp property. Since the streams are stored as a series of NaradaBrokering events rather than large video files, users can ask to replay any part of the stream without loading all related events. Live chat is also implemented to improve the real time communication in the system.

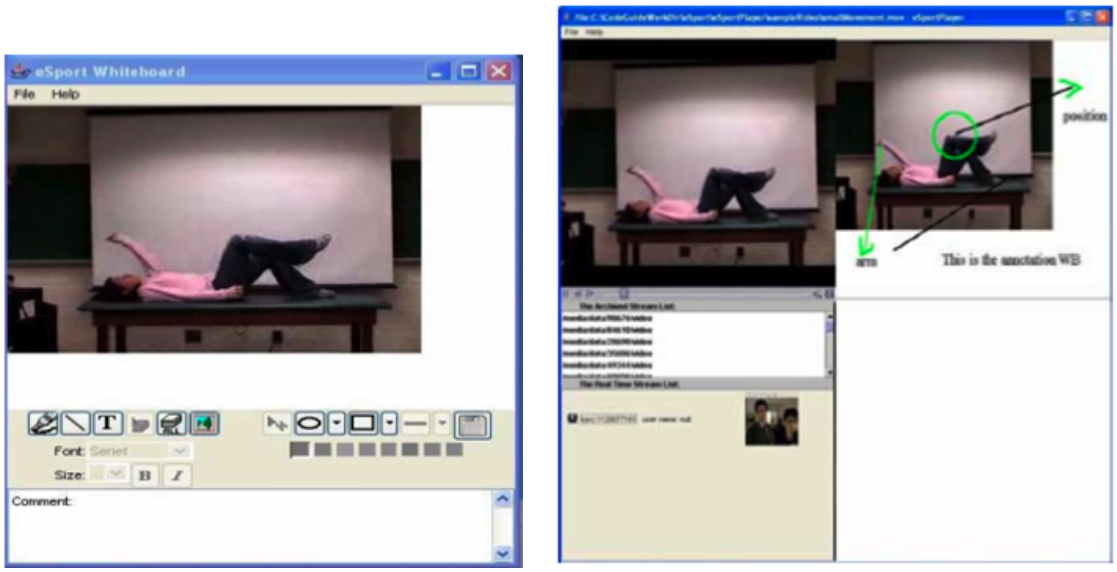

Figure 2.7: Multimedia annotation in eSports

2.1.8 Summary

There are many other platforms/tools that share similar features of above systems. To address all the objectives/problems of building a scalable framework for the collaborative annotation of real time data streams, it is quite obvious that the proposed research should have following major features:

- The target system should be able to support creating, archiving and replaying multiple forms of annotations on either real time or prerecorded data streams without knowing their characteristics.

- The target system should support both synchronous and asynchronous communications on both annotations and content streams.
- A robust session management is required to make the proposed system tolerant to possible hardware or network failures.
- The target system should be adaptive to underlying networks, from high speed Internet to unstable ad-hoc mobile networks.

2.2 Mobile Collaborative Annotation

Researches have been done on different aspects of collaborative annotation on mobile phones. Due to the multimedia capability and location awareness of those devices, most of the research mainly focuses on annotation on digital contents such as photos, audiovisual data and location information. In this section, we give brief analysis on some of these systems and explain how they affect our design.

2.2.1 Mobile Annotation Systems

Many efforts have been spent in bringing multimedia annotation on current mobile platform. In [*Yeh et al.*, 2004], Yeh introduces a hybrid searching technique for location recognition based on image and keywords. It however does not support operations in real time. Reference [*Anguera et al.*, 2008] is an annotation system for digital contents on cell phone but its lack of server side supports makes it impossible for users to collaborate with each other. As an improvement of existing mobile search systems such as Layar[*Layar*, 2012], [*El-Saban et al.*, 2011] uses image and video information in extracting information about a scene. It also associates tags with the content for later usage and supports capturing short videos instead of images in Google Goggle[*Google Inc*, 2010].

In [*Wilhelm et al.*, 2004], Anita et al develops a lightweight client application which uses camera phones to capture images and annotate on them. All the annotation information is stored remotely on a dedicated metadata server and organized in a faceted classification structure. This enables rich description of the images and overcomes the limitations of strictly hierarchical metadata structures and keyword based approaches in prior image annotation systems. However, the limited screen size of the mobile device causes a problem for the system to display and enable navigation on such faceted meta data structure.

Jintao et al investigate four major techniques in their paper [*Wang and Canny*, 2006] for collecting end-user place annotations interactively using cell phones. Based on their usability test results, they conclude that "photo memo plus off line editing" is the most favorite approach in ease of use. Although their approach elaborates on providing most convenient user interface for the end users to generate location based annotation data on images stored on their cell phone, annotation in a team oriented fashion was not addressed.

Most of previously described systems mainly focus on utilizing the ubiquitous feature of mobile devices and their network access with geographic information to support user friendly annotation experience. Few of them have talked about supporting collaboration between mobile and desktop users and almost none of them can support such capability. This becomes the motive of design and implementing the mobile extension of the collaborative annotation framework.

2.2.2 Android Based Annotation Systems

Android is one of the youngest and most promising operating system of the mobile OS family. It is maintained by the Open Handset Alliance [*Google Inc*, 2009] led by Google and it has greatly evolved since 2009. Four major versions have been released in the past three years and there are many android-enabled mobile devices such as

smartphones and tablets currently available in the market. As a descendant of Linux, android supports almost every feature of a modem computer and its user interface is designed to be compatible with all user interactions on regular computers except that they are touch based. The android development framework is inspired and designed based on Oracles Java and swing toolkit which makes it easy to port existing java based system onto the android platform.

In [*Wang et al.*, 2011], Zixuan Wang et al present an image annotation system based on android devices and a dedicated web server. It basically uses the android smart phone to capture images and create tags on specific portions of them. Relevant annotation of the same object on different images is grouped together based on similarity algorithms to help the user share better semantic understanding of the object. Most of the analysis is done by the web server after end users submit their photo tags and android devices are merely working as input devices. Moreover, it is quite difficult for this system to support collaborations on their images in the real time without sending queries to the web server.

CHAPTER III

A Scalable Framework of the Collaborative Annotation

3.1 Architecture Choices

To achieve a robust and flexible collaborative annotation framework, we summarized and compared popular architecture choices made by previously introduced systems based on various of our requirements.

Client and Server(C/S) paradigm was mostly used among early systems such as MRAS and VideoAnnEx. Smart client talks to the server for data that need to be displayed and commits back user inputs for permanent changes. It is the server's responsibility to understand both content and interaction data as well as the logic of binding them together. Client code simply interprets the relationship and presents it to the end user. This makes it easy to extend the system to different platforms if the data presentation and user interaction are simple and universal. However, with the increase of data throughput and complexity of the meta data, this centralized server becomes a bottleneck of the system and it becomes almost impossible to be extended. It normally ends up with deploying duplicate servers and having a dedicated task scheduling server to load balancing the task. User interactions on different platforms such as desktops and mobile devices may also be quite different from each other. Sometimes it is hard to port client codes from one platform to another.

Multi-tier Architecture tries to resolve above problems by encapsulate and inte-

grate various functionalities in different tiers. It is a kind of client-server architecture in which the presentation, the application processing, and the data management are logically separate processes. The simplest known multi-layer architecture is 2-tier or client/server system. This traditional two-tier, client/server model requires clustering and disaster recovery to ensure resiliency. While the use of fewer nodes in an enterprise simplifies manageability, change management is difficult as it requires servers to be taken offline for repair, upgrading, and new application deployments. Moreover, the deployment of new applications and enhancements is complex and time consuming in fat-client environments, resulting in reduced availability. A 3-tier information system consists of the following layers:

- Presentation layer presents information to external entities and allows them to interact with the system by submitting operations and getting responses;
- Business/Application logic layer or the middle-ware programs that implement the actual operations requested by the client through the presentation layer. The middle-tier can also control user authentication, access to resources as well as performing some of the query processing for the client, thus removing some of the load from the database servers;
- Resource management layer also known as data layer, deals with and implements the different data sources of an information system.

In fact, a 3-tier system is an extension of a 2-tier architecture where the application logic is separated from the resource management layer[*Edwards*, 1999]. By the later 1990s, as the Internet became an important part of many applications, the industry extended the three-tier model to an N-tier approach. As a consequence the data tier became split into a data storage tier and a data access tier. In very sophisticated systems an additional wrapper tier can be added to unify data access to both databases and web services.

Service Oriented Architecture (SOA) is about how to design a software system that makes use of services of new or legacy applications through their published or discoverable interfaces. Systems are often distributed over networks. SOA also aims to make services interoperability extensible and effective. It prompts architecture styles such as loose coupling, published interfaces, and a standard communication model in order to support this goal. Systems such as eSport and SIDGrd choose this architecture to overcome problems in C/S paradigm. Their system capabilities are delivered and consumed via loosely coupled, reusable, coarse-grained, discoverable, and self-contained services interacting via a message-based communication model(Naradabrokering in eSports and SOAP/RESTFul messages in SIDGrid). Unlike C/S model, which is based on design and development of tightly-coupled components for processes within an organization, using different protocols and technologies such as CORBA, DCOM, etc, SOA focuses on loosely-coupled software applications running across different administrative domains, based on common protocols and technologies, such as HTTP and XML. SOA is related to early effort on the architecture style of large scale distributed systems.

Based the analysis and comparison of above architectures, SOA is obviously the better choice for the collaborative annotation framework to provide high throughput streaming services such as archiving and replaying as well as robust session managements. It also helps the framework become scalable by providing different levels of servicing quality and capability. We also choose the idea of functionality separation and encapsulating in multi-tier architecture to make sharable components on the client side to be reused on different platforms easily. More details can be found in the following chapters.

3.2 Messaging Systems

There are several useful standards in this field. The best known is the Java Message Service (JMS)[*Oracle*, 2001] which specifies a set of interfaces outlining the communication semantics in pub/sub and queuing systems. Advanced Message Queuing Protocol (AMQP) [*amqp*, 2012] specifies the set of wire formats for communications; unlike APIs, wire-formats are cross platform. In the Web service arena there are competing standards WS-Eventing and WS-Notification but neither has developed a strong following. We now present in Table 3.8, a comparison between a few common messaging and queuing systems.

We also give MuleMQ[*Mule Soft*, 2010b] which is the messaging framework underlying the enterprise service bus (ESB) [*Mule Soft*, 2010a] system Mule developed in Java of which there are 2500 product deployments as of 2010. The focus of Mule is to simplify the integration of existing systems developed using JMS, Web Services, SOAP, JDBC, and traditional HTTP. Protocols supported within Mule include POP, IMAP, FTP, RMI, SOAP, SSL, SMTP. ActiveMQ [*Apache*, 2007] is a popular Apache open source message broker while WebSphereMQ[*IBM*, 2002] is IBMs enterprise message bus offering. Finally we list the open source NaradaBrokering[*Pallickara and Fox*, 2003] that is notable for the broad range of supported transports and was successfully used to support a software MCU (Multipoint Control Unit) for multi-point video conferencing and other collaboration capabilities.

System Features	Active MQ	Mule MQ	Websphere MQ	NaradaBrokering
JMS compliant	Yes	Yes	Yes	Yes
Distributed broker nodes	Yes	Yes	Yes	Yes
Delivery guarantees	Based on journaling and JDBC drivers to databases.	Based on the Realm disk store, 1 file per channel, messages purged by TTL.	Exactly once delivery supported	Guaranteed and exactly-once
Ordering guarantees	Publisher order guarantee	Not clear	Publisher order guarantee	Publisher-order, time-order based on Network Time Protocol
Access Model	Using JMS classes	JMS, Adm. API, and JNDI	Message Queue Interface (MQI), JMS	JMS, WS-Eventing, and native interfaces
Support for buffering	Yes	Yes	Yes	Yes
Time decoupled delivery	Yes	Yes	Yes	Yes
Security scheme	Authorization based on JAAS for authentication	Access control lists, authentication, SSL for communication	SSL, end-to-end application level data security	SSL, end-to-end application level data security, and ACLs
Support for Web Services	REST	REST	REST, SOAP based interactions	WS-Eventing
Transports	TCP, UDP, SSL, HTTP/S, Multicast, in-VM, JXTA	Mule ESB supports TCP, UDP, RMI, SSL, SMTP and FTP	TCP, UDP, Multicast, SSL, HTTP/S	TCP, Parallel TCP, UDP, Multicast, SSL, HTTP/S, IPSec, non-blocking TCP
Subscription formats	JMS spec allows for SQL selectors, Also access to individual queues.	JMS spec allows for SQL selectors, Also access to individual queues.	JMS spec allows for SQL selectors, Also access to individual queues.	SQL Selectors, Regular expressions, ¡tag, value¿ pairs, XQuery and XPath

Table 3.1: Comparison of Selected Messaging and Queuing Systems

Note that the four non-cloud systems support Java Message Service JMS. Also

there are some key features of messaging systems that are listed in the table but not discussed in this brief section. These are security approach and guarantees and mechanisms for message delivery. Time decoupled delivery refers to the situations where the producer and consumer do not have to be present at the same time to exchange messages. Fault tolerance is also an important property some messaging systems such as NaradaBrokering can back up messages and provide definitive guarantees. This table is only illustrative and there are many other important messaging systems. For example RabbitMQ[*VMWare*, 2010] is a new impressive system based on AMQP standard.

It is quite obvious from the table 3.1 above that NaradaBrokering provides complete and diverse feature supports and meets our requirements for delivering a robust and mature collaborative platform. And you can find from the following sections that we have taken full advantage of this messaging system to achieve fluent real time communications between clients on both desktop and mobile platforms.

3.3 Architecture of the Framework

Figure 3.1 below depicts a typical scenario of using our framework. A stream annotator is feeding a live video stream to the system and making notes on it. Client A and B are live collaborators in the same session and they are able to ask questions on the video stream while it is being played. Another client using a handheld device is watching the collaboration activities between the annotator and client A and B. Session information, annotations and stream data are transmitted and exchanged using Naradabrokering events. All events are automatically stored into the stream repository for later replays. Different metadata are stored in each events header, and information within them facilitates functions such as stream synchronization and system recovery.

There are three major components in the system: Session Manager, Annotation

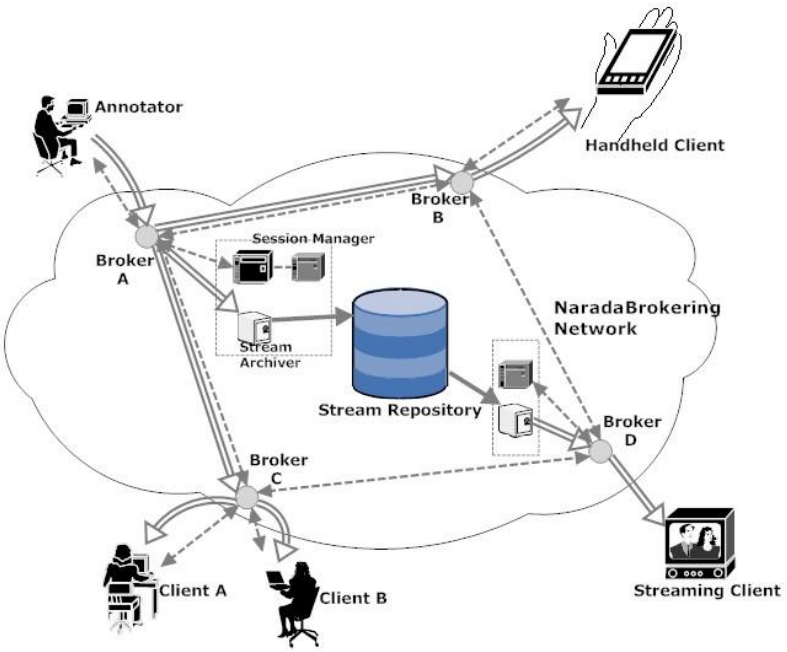

Figure 3.1: Detailed system architecture

Client and Stream Archiver. Session Manager maintains all session related information such as client joining or leaving. The client is responsible for generating content streams as well as receiving and replaying streams from other clients. It also parses annotation events to reproduce actual annotations on the content stream. Stream Archiver is spawned by Session Manager to archive live streams in the stream repository, either locally or remotely. It is also responsible for retrieving archived streams as per the clients requests.

3.4 Session Management

Due to the pub/sub nature of the Naradabrokering[*Pallickara and Fox*, 2003] system, we use heartbeats to manage the session information in the system. Each component in the system continuously publishes its own heartbeat event to public channels. All clients will monitor heartbeat events in the session channel and maintain their own copies of the session status, i.e. list of active clients in current session. Unresponsive clients will be removed from the list if other clients cannot hear from

them for more than three seconds. Session Manager monitors the session channel as well and periodically broadcasts its own client list as the standard for participating clients to synchronize their lists with. Session Manager will also monitor the service channel to control active stream archivers and remove unnecessary ones. A status report will be generated and stored in the local file system and remote stream repository after a customizable period of time.

As the core management component of a distributed system, Session Manager should be available all the time and be able to recover from disastrous situations such as program crashes and power outages. We use two strategies to maintain such durability: Local recovery and Remote recovery.

- Local recovery: Alongside the running Session Manager, a daemon process (gray manager in Figure 1) keeps collecting session information as other clients do. It starts taking over the management responsibility when the running manager freezes and stops publishing standard heartbeat. It will kill the original manager process, changes its own status by parsing the latest status report on the file system and create another daemon process to take over its previous job. Since clients will not check the source of the standard heartbeat, they will not know the manager has been replaced.

- Remote recovery: We could not apply local recovery if there were hardware problems or power outages on the running manager machine. In such circumstances, all clients will find a best machine among them by exchanging and comparing their hardware information. The most appropriate client will create the manager process, adjust its status according to the remote status report and start collecting information from both the session and service channels.

3.5 User Experience Design

One important feature of this collaborative annotation framework is to separate its graphic user interfaces from underlying service components such as annotation archiving, distribution and replaying. Through such separation, user experience researchers can focus on investigating capabilities and differences between targeted supporting platforms and design best user interfaces. In general, a system based on this annotation framework will need to provide following experiences on its client:

- A session chooser which users can select a session of their interests to join.
- A stream list which shows what content/archived streams are available for play within the session.
- An annotation panel for displaying/rendering streams so that users can interact with.
- A stream replay panel(Optional) which helps users to find streams that they have opened and annotated.
- A time line panel(Optional) which enables users to select arbitrary time spot of the opened/replaying stream so that they can add annotations at.

There can be other optional user experience elements such as stream preview window/panel as well as system/session monitor. They are defined by the system requirements and how to design and deliver them for various platforms is out of the scope of this dissertation. In section 4.1 of next chapter, we give a detailed explanation on how we separate the user interface from service components through a layered architecture for presentation, logic and data transmission. User researchers can focus on good presentation designs without concerns about breaking underlying functionalities if the system complies to the communication interfaces defined in the framework.

3.5.1 A Sample Desktop User Interface on Windows

Figure 3.2 is a snapshot of our sample annotation client running on Windows. We implement the client using SWT library[*Eclipse*, 2012], an OS-independent widget toolkit from the Eclipse project. The client comprises a tree based client list and three composite panels. Each panel can be maximized to show as much information as possible.

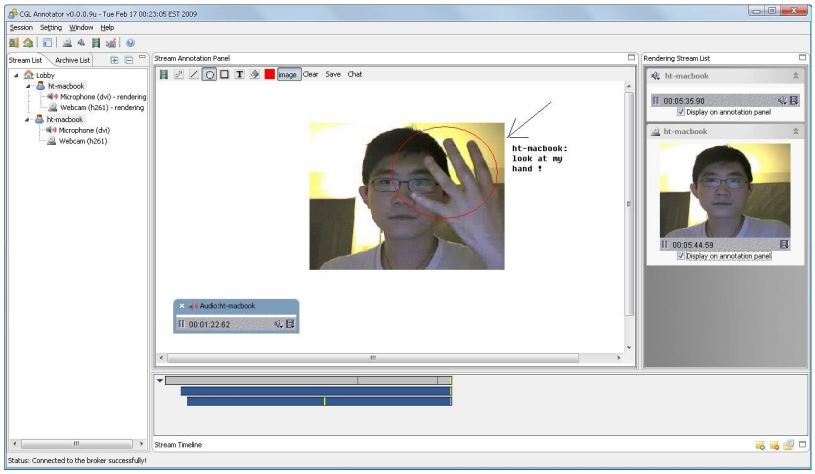

Figure 3.2: A Snapshot of The Sample Desktop User Interface.

The client list on the left displays all participating clients in the same session. The user can open any data stream (video steam in the snapshot) being sent by a client. Once the receiver of this data stream is created and started successfully, the renderer window will be displayed in the stream renderer list on the right panel. Users can also select to create a clone of the playing renderer to the center panel by checking the check box underneath it. A stream progress widget is also created on the progress panel below once the clone starts playing. Unlike the original renderer window on the right, the cloned renderer can be positioned anywhere on the center panel and the user is able to either rewind or fast forward the playing content by dragging the progress indicator on its stream progress widget.

Alongside the client list, there is an archive list that only displays information of data streams stored by stream archivers. Users can apply all available operations on

these archived streams as if they were normal live streams. There is no difference between them and the live stream since they are just duplicates of the stored live streams from the event repository, loaded and published by stream archivers. More details of archiving and replaying streams will be explained in the next section.

3.5.2 User Interfaces for other Platforms

Due to the cross platform supports of SWT and JMF, our annotation client for Windows can be used directly on other desktop platforms such Mac OS and Linux. The only limitation is that JMF may not have full audiovisual encoding/decoding support on these platforms. This is not a limitation caused by the design of the collaborative annotation framework and can be easily solved by replacing the media component with other modern technologies.

3.6 Structure and Feature Comparison

As a descendant of GlobalMMCS [*Wu et al.*, 2006], NaradaBrokering [*Pallickara and Fox*, 2003] and other collaboration systems [*Bulut*, 2007] built in Pervasive Technology Insitute, this collaborative annotation framework greatly enhances functionality and makes their feature components such as reliable message delivery, stream recording/archiving and media playback reusable to its users, improve the system robustness and avoid duplicate work among derived annotation systems that share similar characteristics. The following table 3.2 gives a structure and feature comparison between the annotation framework and those previous systems.

Features	**GlobalMMCS**	**High Performance Stream Recording and Manipulation System [*Bulut*, 2007]**	**Collaborative Annotation Framework**
System Architecture	C/S, extension through gateways	C/S, extension through gateways	Decoupled, Service Oriented
Cross OS support	Windows/Linux	Windows/Linux	Windows/Linux/MacOS(Limited, see table 4.1) /Android
Multimedia Support	support from JMF	support from JMF	support from JMF and Android Media Framework
Generic Stream Support	Multimedia Only	Yes, need translation gateways	Yes, both client and service node
Reliable Data Delivery	No	Yes, extension to the NaradaBrokering Reliabile Delivery Service	Yes, extension to the NaradaBrokering Reliabile Delivery Service
Mobile support	Basic web based thin client	Basic web based thin client	Full featured native mobile client
Robustness	Unstable, demo only	Unstable	Stable with correctly deployed services

Table 3.2: Comparison with Previous Systems

As you can see from the table, previous systems such as GlobalMMCS were implemented based on a centralized design which makes them difficult to be extended and scale. Various dedicated gateways were introduced to extend their supports for other platforms. This increased the complexity of the service components on the server side since they have to detect the payload type through multiple handshake communications between the client and server, set up correct extension gateways to service the incoming data stream and translate/transcode them into system formats that service components can understand. A single failure at each step explained here may cause the server end up in a unstable state and crash eventually.

Through a service oriented design (section 3.3), the collaborative annotation framework introduced in this dissertation avoids above problems by decoupling feature components into separate services that can be started/stopped based on requests,

using a simplified yet robust session management scheme (section 3.4) to minimize the system complexity and adding extra redundancy (section 6.2.3) to the underlying messaging system to ensure service availability. This service/plugin based design also enables us to introduce new features such as generic stream support and mobiles support. We give a complete list of them as follows:

1. Multimedia support: enables recording, rendering and archiving audiovisual data streams through Java Media Framework [*Oracle*, 2004] for desktop machines and Android Media Framework[*Open Handset Alliance*, 2009] for mobile devices.

2. Simple Desktop/Mobile Client Templates: simple yet complete client templates for the framework users to reuse on different operating systems.

3. Layered Annotation Programming Interface: a decoupled programing interface that enables framework users to extend it through different recorder and renderer implementations (see section 4.1).

4. Decoupled service oriented framework: separate service/feature components from user interfaces and underlying messaging transmission.

5. Session Manager: a simple session management service which provides robust recovery strategies(see section 3.4).

6. Stream Archiver: a robust real time data recording and replaying service which enables preserving and playing back annotation meta data as well as their content streams.

7. Annotation Manager: a service that provides indexing and searching within the stored annotation meta data.

8. Reliable Data Delivery and Repository: an extension to the underlying NaraBrokering messaging system that enables reliable stream delivery as well as storing(see chapter VI).

More details for some of these features and services can be found in chapter IV and chapter V.

CHAPTER IV

Annotation on Generic Streams

The most important feature of this research is to provide capabilites of collaborative annotation on generic real time data. In this section, we will address some research issues we encountered during the design and implementation of the desktop interface of the collaborative annotation platform. These issues vary from simple integration interface for the platform users, Stream rendering and archiving and annotation management.

4.1 Annotation Interface

Figure 4.1 below shows three layers of our annotation interface: Transmission layer, Logic layer and Presentation Layer from the bottom up. Each takes its own responsibility of processing the streaming data.

The Transmission layer is responsible for creating and managing actual data transmission handlers (called DataTransmitter in the source). Each transmission handler contains a pair of NaradaBrokering event consumer and publisher, and it subscribes itself to a particular topic specified by the ID of the stream it operates on. In order to minimize the cost of handler creation and termination, a pool of handlers (around 5 handlers) are created during the start up of the client. Similar to the Java thread pool, transmitting handlers are assigned and recollected by a handler manager.

The Logic layer works as an important mediating layer between the Transmission layer and the Presentation Layer. For stream capturing and rendering, a stream

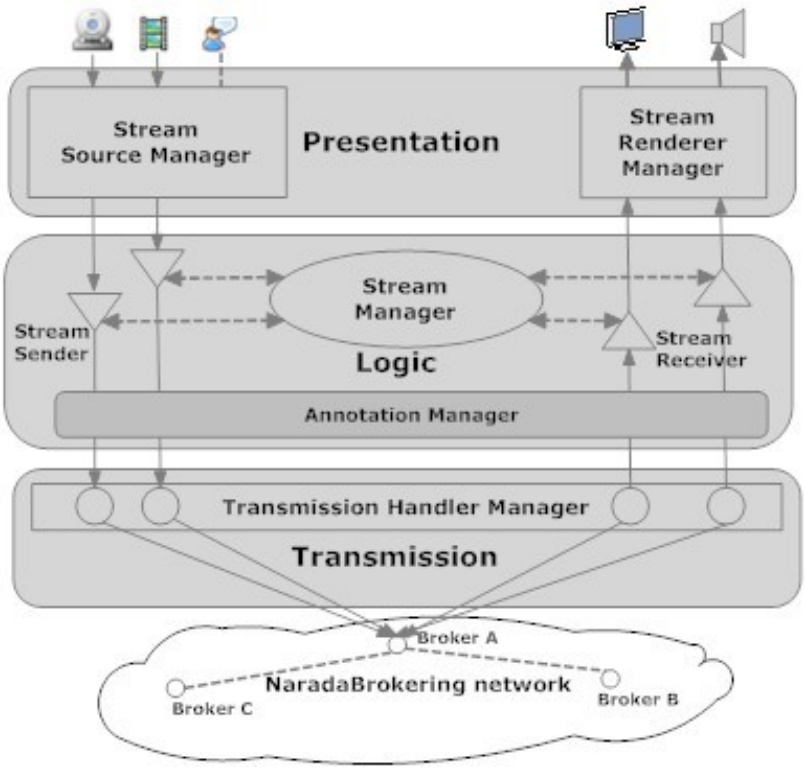

Figure 4.1: Three Layers of the Annotation Client Interface

sender or receiver will be created to connect a stream source/renderer from the presentation layer with a transmitting handler from the transmission layer and start the processing. There is a stream manager in this layer to manage all active senders and receivers. The Annotation manager also sits within this layer to associate and synchronize content data streams with the annotation streams.

The Presentation layer is the upper-most layer and it contains the graphic user interface, stream source and renderer managers. Similar to the DataSource class in the JMF library, a stream source is an object that can generate real time data constantly when it is started. It can be paused or stopped. Stream renderers are used to decode received stream data and display the content on the screen.

Figure 4.2 below is a class diagram that shows the interrelationships between the stream source/renderer interfaces and the stream sender/receiver classes.

Since the stream source/sink interfaces in above picture only define the generic

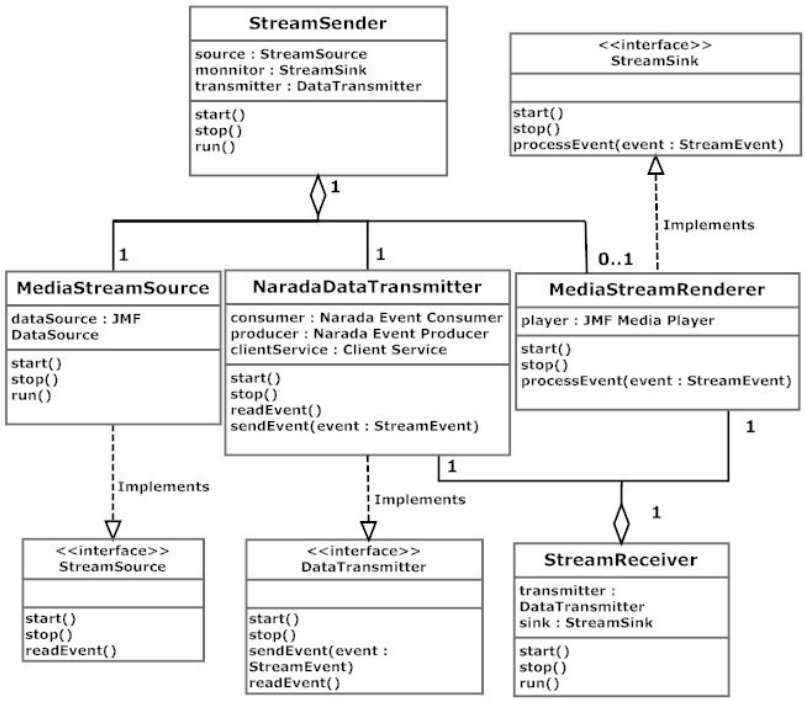

Figure 4.2: Class Diagrams of Stream Processing Interfaces.

behaviors of a real time data stream, users can easily write their own stream sources and renderers to extend the system. They just need to implement those interface methods in their existing source/rendering classes and compile them with the client source. This will save a lot of effort as opposed to understanding and modifying source codes of the entire system. In our current release, we have implemented several stream sources such as video/audio capturing source, file capturing source and screen capturing source and their corresponding renderers. With the help of the GlobalMMCS[*Wu et al.*, 2006] media module, our system supports various video/audio formats on different operation systems. They are listed in the Table 4.1 below.

OS	Video	Audio	Screen Capture
Windows	H.261, H.263, DIVX, JPEG	ULAW, GSM, DVI, G729	H.261, DIVX, JPEG
Linux	H.261, H.263, JPEG	ULAW, GSM, DVI	N/A
Mac	H.261, JPEG	ULAW, GSM, DVI	N/A
Android	H.263, H.264	ULAW, GSM, G729	N/A

Table 4.1: Supported Multimedia Formats

4.2 Stream Rendering

There are two modes of rendering received data streams in our client: live and buffered. The first mode is the default one. Events of an incoming data stream are temporarily stored in a small in-memory buffer to reduce the influence of possible event losses in the transmission. Sometimes, it would be useful if users could rewind the playing content to the exact position that they want to insert annotations at. This requires enabling the buffered mode of rendering the stream. As depicted in the following figure 4.3, decoded video frames are written into a temporary file and can be retrieved from any time spot based on the frame rate information inside the stream?s video codec. When the user makes a rewind operation on the current stream progress, a buffered stream source is created at the correct playing time and started to read the correct video frames from the buffer file for the stream renderer to display. A reading clock controls the speed of the buffered source and makes sure that it generates frames at the right frame rate. Despite the disk access overhead introduced here, this feature enables annotation on live video streams while they are being watched.

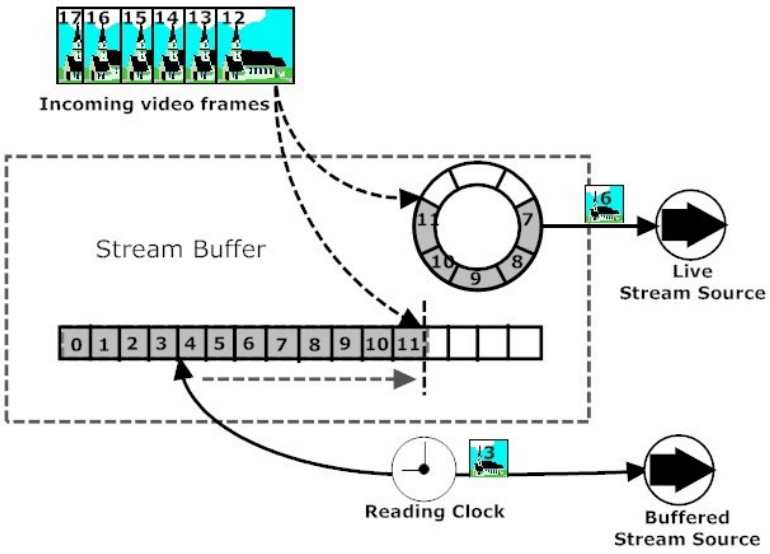

Figure 4.3: A Running Example of the Stream Buffer

4.3 Stream Archiver

Stream Archiver is one of the most important components in the system. It takes the responsibilities of archiving live data streams and replaying them per the client?s requests. In our current implementation, the archiver stores every stream event into a remote database alongside the meta-information such as time stamp and stream description in the event?s header. When a request of replaying a particular data stream is received, the corresponding archiver will read all stream events based on time range information within the request. Events will be published to a specific replaying topic based on the request ID known by the requesting client.

As explained in the previous section, Stream Archiver is monitored and controlled by the Stream Manager. When a sending stream is stopped, Stream Manager will terminate its corresponding archiver unless there are some clients requesting to replay this stream.

4.4 Annotation Management

In Figure 4.4, you can see that there is a stream progress panel on the bottom of the client. It allows users to control the rendering of data streams on the center annotation panel and create annotations on them. The stream progress widget displays the length and playing progress of the stream. When an annotation is created, information of all the stream renderers on the annotation panel is stored into a XML DOM object and each renderer starts to update this object with its newest progress. Following is an XML example generated from a simple annotation DOM object.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Streams>
  - <Stream>
      <ID>dfc3796d-a536-47ad-a25a-b8c3961d2179</ID>
      <Type>Text</Type>
      <Start>1233695738712</Start>
      <Duration>98359</Duration>
    </Stream>
  - <Stream>
      <ID>4374cb48-ea94-4c0e-8f18-0b93b96026db</ID>
      <Type>Audio</Type>
      <X>0</X>
      <Y>0</Y>
      <Start>1233695788790</Start>
      <Duration>48281</Duration>
    </Stream>
  - <Stream>
      <ID>eae834f3-dd73-42eb-9b77-8223de01b469</ID>
      <Type>Video</Type>
      <X>254</X>
      <Y>148</Y>
      <Start>1233695783821</Start>
      <Duration>53282</Duration>
    </Stream>
</Streams>
```

Figure 4.4: Annotation DOM Object in plain XML

As seen in the above picture, there are no actual stream events stored in this XML file. We only record information that represents the layout of all active streams in the annotation panel, for example, position of the renderer on the center annotation panel, absolute start time of the stream and its duration. All this information will be used to reconstruct the annotation scenario later on.

When the annotation owner closes the annotation, an XML copy of the annotation object will be saved remotely in the annotation storage. A local copy is also created as backup for fast accessing. When the user decides to replay the annotation he creates, the client will first check the local file system before asking the remote repository. The Dom object will be parsed and created from the XML file and all renderers will be regenerated as well as their annotation.

CHAPTER V

Annotations in the Mobile Environment

From the introduction and analysis of existing mobile annotation systems in section 2.2, we can see that most existing systems are limited to capturing simple digital data such as images and geographic data. And the annotation methods that these systems support are also quite primitive and restricted to simple tagging and text comments.

Since we want to provide similar user experience in the mobile extension as in the desktop client of our collaborative annotation framework, it is important that the mobile extension should be able to support several key features required by the system. We give a side by side feature support comparison between technologies that android provides and those we used in the collaborative annotation framework in Table 5.1. It is quite obvious that the android platform meets almost every requirement to build such an extension for the collaborative annotation framework. And as mentioned in the previous section 2.2, the java based android development framework also makes migrating key components of the existing annotation framework into the mobile environment quite simple.

Features	**Annotation Framework**	**Android Platform**
User interface	GWT, AWT	Android UI framework
Audiovisual Capturing	JMF based	Android Multimedia Framework
Image Process	GWT, AWT	OpenGL ES 1.0/2.0
Whiteboard	GWT based canvas	Android Canvas
Data Transmit	NaradaBrokering, RabbitMQ	Simple RTSP streaming
Data Storage	Raw data file, XML metadata	Raw data file or xml file
Location sensor	3rd party sensors	Supported by default

Table 5.1: Comparison of Technologies Used to Support Different Features

5.1 Collaboration bewtween mobile and desktop clients

Figure 5.1 below shows how the collaborative annotation is done between desktop and mobile users in the system. Both content and session streams are transmitted through NaradaBrokering network. Each time a mobile user logs on the system, it will firstly send out a query event to request latest session information. Once it receives such an update from the session manager, it will subscribe to the corresponding topic and start the underlying broker client to receive data streams that its user chooses to process. If the selected data stream is a multimedia data stream that requires streaming support of android platform, a Stream proxy will be created to redirect the payload of NaradaBrokering events for the android media player to render locally. The design of the proxy will be explained in details in section 4.2. Events for other types of data streams will be passed on to corresponding handlers the same way as in the desktop client.

There are two major differences between the mobile client and the regular desktop clients in the above picture. Since the mobile users are more prone to network issues

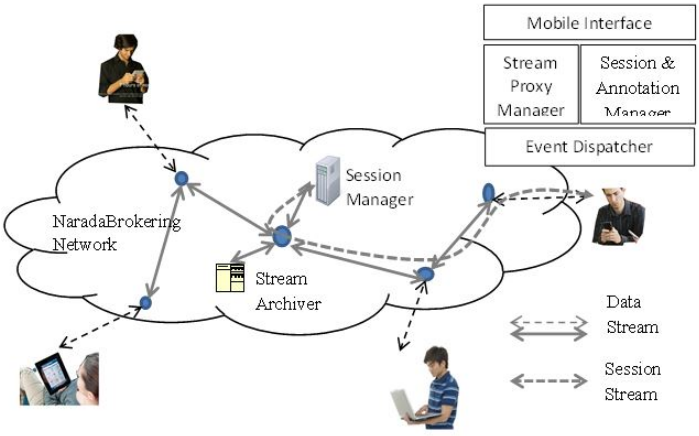

Figure 5.1: Collaborative Annotation between Desktop and Mobile users.

such as disconnection and low connectivity, we need to design a better mechanism for them to save and restore their session status from possible connection problems. Due to the lack of direct RTP support for android media players [19], we need a proxy to understand RTSP requests from the android media player and feed it with the raw RTP data from NaradaBrokering events.

5.2 Improved session control for the mobile environment

In our collaborative annotation platform, we use heartbeats to manage the session information due to the pub/sub nature of the NaradaBrokering platform. Each component in the system continuously publishes its own heartbeat event to public channels. All clients will monitor heartbeat events in the session channel and maintain their own copies of the session status. Unresponsive clients will be removed from the list if other clients cannot hear from them for more than several seconds. In our framework, a dedicated Session Manager as in Fig. 1 is running and responsible for monitoring the session status and synchronizing with every client nodes. It will also generate status reports periodically and store them in the remote storage node. Since the mobile client may reside in low bandwidth networks and has higher probability of losing connection with the system, we decide to make our mobile client synchronize

the session monitor only and ignore heartbeat events from other clients to reduce the possibility of misjudging their status. And if the mobile client detects its session information is stale, it will send out a request for a batch update since last successful synchronization. Figure 5.2 below depicts the procedure that our session monitor handles abnormal leaves of mobile clients.

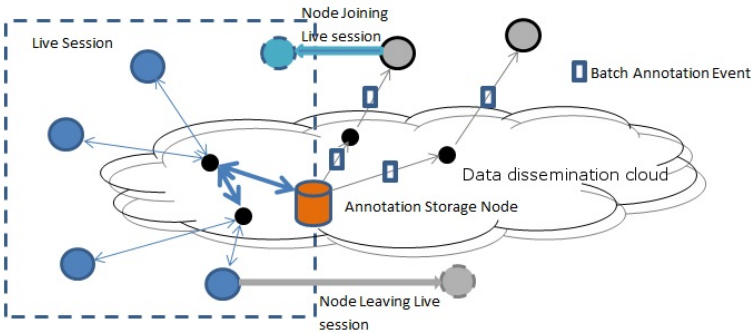

Figure 5.2: Methods of annotation distribution

From the picture above, we can see that once a mobile client rejoins the same live session in our system, it will compare its status with the session manager based on the timestamp and send out batch update request if necessary. The session communication has been minimized to the lowest necessary level to reduce the possibility of status misjudging due to high possibility of network outages. Besides the session information, other metadata like past annotation events and stream changes are also included inside the batch update events.

5.3 Multimedia Proxy

Since the android multimedia framework doesnt support direct RTP streaming, we design and implement a multimedia proxy to communicate with the android media player and redirect to it actual RTP media packets. This proxy is basically a simple RTSP server which handles requests from the android media player for media playback and codec information. After a successful communication, the proxy feeds in the

player RTP packets extracted from the NaradaBrokering events. Figure 5.3 below shows how the mobile client processes multimedia data streams sending by desktop clients.

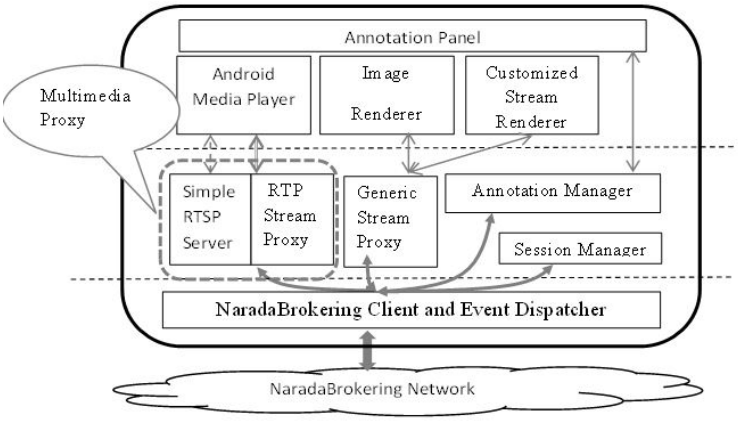

Figure 5.3: Multimedia proxy for audiovisual stream playback.

When the mobile user chooses to play a multimedia stream from the live stream list, a proxy will be created to receive NaradaBrokering events from the corresponding topic, extract the RTP packets from their payloads and buffer them locally. An android media player will then be created on the mobile client and sends out a RTSP request to the proxy for media data for playback. Once all the RTSP based communications such as OPTIONS, DESCRIBE, SETUP and PLAY are done successfully, the media player will start to receive RTP packets from a local port and play the media content defined by the proxy. Currently only H.263 format for the video and Mpeg-3 format for the audio are supported due to the limitation of the android multimedia framework [20]. Other media codecs such as H.261 and Divx/Mpeg4 which are supported on desktop clients are not available on the mobile client currently. The local buffering and communication between the proxy and the media player can cause a delay of initial playing of the audiovisual data. But once the initialization is finished, fluent collaborative annotations on media contents are achievable on a reasonable level within high speed networks. Our preliminary experiment has proved this in the

latter section.

Figure 5.4 below comprises snapshots of the mobile client running on an android smart phone. It is made up of two major activities (running entities on the android platform) that are responsible for session/stream selection and stream annotation. Image on the left shows a demo session that contains multiple attendees. One of them is sending out a live video stream as well as a live audio stream. Once the client user selects to open a video stream from the list, an annotation activity will be brought up as shown in the bottom right picture. Annotation operations are available for selection on a top floating tool bar and they can be hidden to provide better view of the streaming content if necessary. An extra annotation panel can be brought up from right to display past annotations generated by other clients. If the mobile user selects to view one of those annotations, the media player will be rewound to the correct time spot based on the time stamp information of the annotation. The annotation itself will be layout on top of the content stream. Most annotation operations available on desktop clients are also implemented on the mobile client in order to maintain the same user experience within the system.

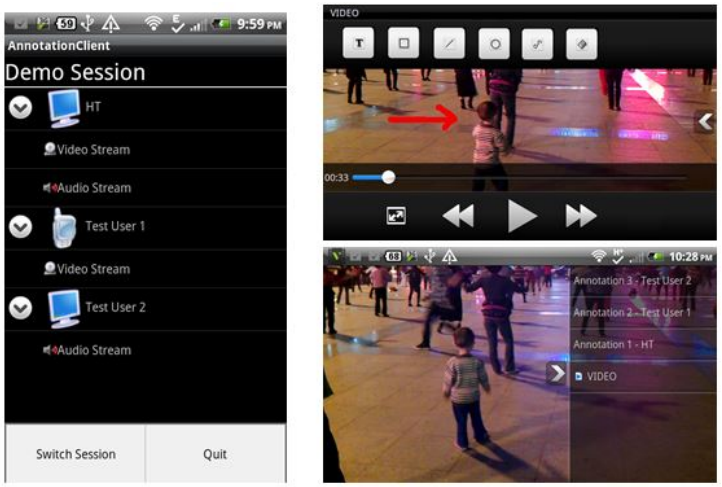

Figure 5.4: Annotation interface of the mobile client.

This client can provide better user experience on android based tablet devices due to larger screens and more accurate user interactions. The layout of components in

above images may change slightly but the actions would remain the same on both mobile and tablet devices.

5.4 Adapting annotation meta-data

As described in section 4.4, we use XML DOM objects to save information that represents the layout of content streams in the annotation panel and related annotations. Due to the limited display size of the mobile client, we make changes to the schema of the XML metadata by adding types of source device, stream source location and so on. Annotation events created by mobile clients will also contain geographic information with them for future features such as annotation search and recommendation based on location.

CHAPTER VI

Jitter Reduction and Fault Tolerant Services

To enable its user to deliver robust and high quality collaborative annotation systems, our framework needs to be fault tolerant to both software and hardware abnormalities. In this chapter, we are going to introduces several built in services that we added to ensure the robustness of the framework.

6.1 Jitter Reduction Service

Jitter reduction is considered to be one of the most important quality of service measurements within collaborative systems. It refers to discontinuous playback of the content payload caused by network problems. And it can be controlled in a reasonable level through local buffering. In this section we will discuss on why and how we incorporate a jitter reduction service in the underlying messaging system.

As a typical example of real time systems, multimedia applications mandate timely delivery of content and generally sustain loss of media packets very well. This makes UDP a very good choice for transporting media, since unlike TCP it does not incorporate an error detection/correction mechanism which adds delays associated with individual packets. Since UDP is point to point, it is best to use RTP over UDP when sending streams from one client to another directly. It is generally difficult to guarantee that all data traversing a packet-switched network will experience exactly the same delay. Packets encounter queues in switches or routers and the lengths of these queues vary with time, meaning that the delays tend to vary with time, and

as a consequence, are potentially different for each packet in the A/V stream. In the case of audio streams, high jitter values can cause voice breaks while in the case of video streams high jitters may cause degenerations in the image quality. In order to overcome the negative effects of high jitter, real-time audio/video clients typically have a buffer which buffers events up to 200 milliseconds and then proceeds to release them preserving the time spacing between packets. These buffering and time spacing services enhance jitter reduction. The way to deal with this at the receiver end is to buffer up some amount of data in reserve(as described in section 4.2), thereby always providing a store of packets waiting to be played back at the right time.

As shown in previous chapters, messaging systems such as NaradaBrokering can be used to deliver multimedia content, while brokers are distributed over a wide range of geographical area with different Internet connections. Same high jitter reasons may still exist for messaging systems.

Most messaging systems lack such a mechanism when they are used to deliver multimedia content for collaboration systems, such as transporting multimedia content within event to the client. This requires that a necessary Jitter Reduction Service in messaging systems, especially if collaboration sessions contain replay of archived streams. This Jitter Reduction Service requires that events are timestamped using NTP timestamps explained in the previous chapter. Because only in that case events from different streams generated from different machines would be ordered correctly and timespacing between events preserved.

In order to achieve low jitter, two services are incorporated within the Jitter Reduction Service, one is Buffering Service which orders events and the other is Time Differential Service, which preserve the time space between events when releasing them. Figure 6.1 shows this relationship between these two sub services.

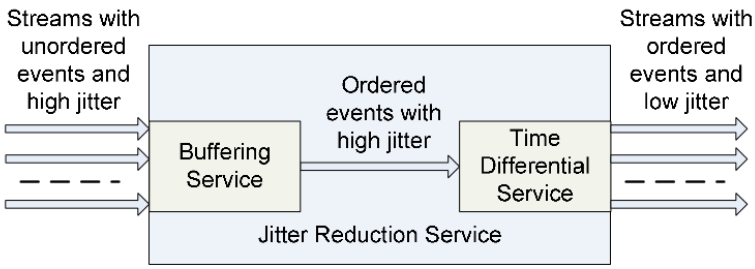

Figure 6.1: Jitter Reduction Service

6.1.1 Time Buffering Service

Buffer size normally varies based on the requirements of the content being playback. In real-time videoconferencing systems, for example, the buffer size of the messaging system is usually set to a low value so that packets will be delayed for up to 200msec while in streaming applications such as Youtube, the buffer size is set to a relatively high value which causes long delays such as 5-10 seconds.

Buffer size also affects the jitter. Small buffer size may cause high jitter while larger buffer size can helps to reduce the jitter. Buffering service described in this section provides a customizable buffer size based on the entitys needs.

If a packet is delayed a long time, it goes into the buffer until its playback time arrives. If it gets delayed for a long time, it will not be stored for very long in the receivers buffer before being played back. Therefore, we have effectively added a constant offset to the playback time of all the packets as a form of insurance. The only time we run into trouble is when packets get delayed in the network for such a long time that they arrive after their playback time, causing the playback buffer to be drained.

The function of buffering service is to buffer some amount of events and release them in an orderly fashion to Time Differential Service. This will ensure that the events received within the buffer duration are ordered and events are delayed for buffer duration.

The design of the buffering service has incorporated four configurable parameters

pertaining to the release of time-stamped messages:

- The first criterion is the number of messages in the buffer maintained by the buffering service. If the number of messages reaches the maximum number of entries, it starts to release the time-ordered messages.

- The second criterion is the total size of the messages in the buffer. This along with the first criterion enables us to circumvent buffer overflows.

- The third criterion corresponds to the time spent by messages within the buffer. In some cases, the rate of messages arriving at an entity may be too slow and this may cause longer and unwanted delays within the buffer. The time-duration factor makes sure that the messages are released after a maximum specified duration if the first two criteria are not met.

- The final criterion is the release factor of the buffer. This typically has a value between 0.5 and 1.0. When any of the release criteria mentioned above is met, it releases at least release_factor * total_buffer_length messages. So that time-ordering for late coming events are achieved to some extend.

6.1.2 Time Differential Service

In collaborative systems simply receiving messages in time-order may not be enough. An entity may also place constraints on the maximum jitter that it can tolerate. The Time Differential Service (TDS) provides two very important functions. First, it reduces the jitter in messages caused by the network. Second, it releases messages while preserving the time spacing between consecutive messages. Preserving time spacing between messages is not an easy task primarily because most operating systems do not provide strict real-time capabilities. Depending on the operating system, the scheduling of processes and threads does not necessarily guarantee the CPU for that process or thread after a specified interval. For example, using

Java on the Windows operating system, user-level threads can obtain the CPU back only after 10 milliseconds. Based on the scheduling configuration of Linux operating system this duration can vary from 1 millisecond to 10 milliseconds or more.

One of the main reasons that TDS uses threads rather than traditional polling to release events in the queue is to avoid high CPU utilizations. In the case of polling, in order to release events in the queue, their timestamps should be checked very frequently. This can lead to very high CPU utilizations. Furthermore, since the rate at which events are generated is not constant: the time spacing between consecutive events vary. Using threads ensures that the CPU utilizations are significantly lower. The reason that we have multiple threads instead of one to release the events in the queue is due to issues related to the programming language (Java) and the operating system. For example, on Linux (Fedora 2), in order to check the timestamps every millisecond, we need to use at least three inter-leaving threads since each thread wakes up after a minimum of 3 milliseconds. On Windows, this value is 10 milliseconds; this high value may not be able to address jitter reduction adequately.

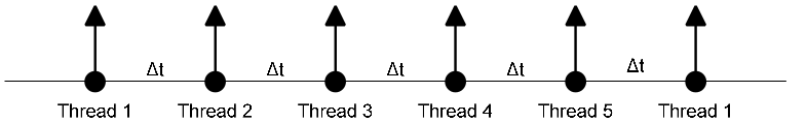

Figure 6.2: Ideal Time Differential Service thread invocation

TDS spawns five threads to process messages released by the buffering service, as shown in Figure 6.2. Threads are configured so that one thread runs for (t time duration and wakes up after 4(t time duration. Since threads are not strict real time (Java threads) we can only approximate the run time and sleep time duration values. Note that TDS itself maintains another buffer for processing. Each thread is initiated one after another with a specified time difference between consecutive initiations. Each thread sleeps for a specified time-slice. By interleaving the durations at which these threads wake-up, TDS can operate on the buffer at finer intervals while ensuring

that CPU utilizations are low. The time-slice interval for individual threads impacts CPU utilization. We have observed that if the time interval between threads is 1 millisecond the CPU utilization stays around 5-6%, when this interval is decreased to 10 microseconds, it can reach about 20-25% on a Linux machine (1.5 GHz CPU 512 MB RAM). When a thread wakes up, it checks to see if any messages need to be released, and it does so if needed. It is done by comparing the messages timestamp, the local clock obtained from the high resolution timer and the time at which the last message was released. By preserving the time-spacing between messages, TDS reduces the jitter significantly.

6.2 Replicated and Fault Tolerant Services

Reliable delivery of messages is an important problem that needs to be addressed in distributed systems. Topics over which authorized publishers and subscribers can have reliable communications are referred to as reliable-topics. The reliable delivery scheme in NaradaBrokering system enables reliable delivery of messages in the presence of link and node failures. The communication links within the system can also be unpredictable, with messages being lost, duplicated or re-ordered in transit over them, en route to the final destinations. This is facilitated by a specialized repository node. In this chapter, after we give an overview of the reliable delivery scheme in NaradaBrokering system, we present our strategy to make this scheme even more failure resilient, by incorporating support for repository redundancy.

6.2.1 Overview of NaradaBrokering Reliable Delivery Service

The scheme for reliable delivery of messages[*Pallickara and Fox*, 2003], issued over a reliable-topic, needs to facilitate error corrections, retransmissions and recovery from failures. In this system, a specialized repository node which manages this reliable-topic plays a crucial role in facilitating this. The repository facilitates reliable delivery

from multiple publishers to multiple subscribers over its set of managed reliable-topics. The only requirement for the basic reliable delivery scheme is that if a repository fails, it should recover within a finite amount of time.

This reliable delivery guarantee holds true in the presence of four distinct conditions.

- Broker and Link Failures: A broker network may have individual or multiple broker and link failures. Once the broker network recovers, the delivery guarantees should be met.

- Prolonged Entity disconnects: If an entity is disconnected and misses events, the delivery guarantee will be met, with the entity receiving all the events missed in the interim once the entity reconnects.

- Stable Storage Failures: It is possible that stable storages present in the system may fail. The delivery guarantees must be satisfied once the storage recovers.

- Unpredictable Links: The events can be lost, duplicated or re-ordered in transit over individual links, en route to the final destinations.

To enable the management of reliable-topics, the repository facilitates the registration and de-registration of authorized clients (publishers & subscribers) for reliable communications over the reliable-topic. Publishers and subscribers that are not explicitly authorized (and registered) by the reliable-topic owner cannot avail reliable communications over that topic. To support error-corrections, retransmissions, and recovery from failures (including those of the repository itself) a repository also needs provision stable storage so that messages and other information pertinent to the reliable delivery algorithm can be stored.

Figure 6.3 summarizes the interactions between entities over a reliable topic. These interactions are explained in the following sub-sections.

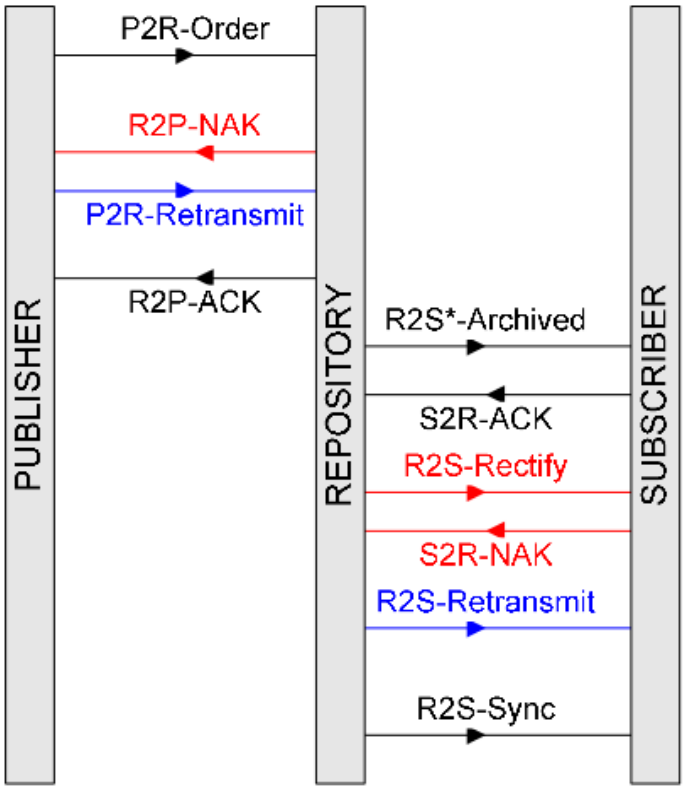

Figure 6.3: Summary of interactions between entities

6.2.1.1 Control-Events

The reliable delivery algorithm involves communications between various entities through the exchange of control- events (summarized in Figure 6.3). The control-events (simply events, for brevity, hereafter) relate to intermediate steps to facilitate reliable delivery, acknowledgements, error-corrections, retransmissions and recovery related operations. Our notation for events identifies the source, the destination(s) and the type of control-event: Source2Destination-ControlType. For purposes of brevity, we use only the starting alphabets of the entities involved in the exchange. Thus, an acknowledgement issued by the repository to the publisher is represented as R2P-ACK. The destination part is in **bold-face** if there are multiple destinations.

6.2.1.2 Publishing Messages

The messages issued by publishers to a topic will also be received at the repository. To ensure that the repository can know about and retrieve missed messages for every published message, the publisher also issues a P2R-Order event to the reliable-topic repository.

A publisher stores every message that it publishes, over a reliable-topic, in its local buffer (maintained in memory), which serves as a temporary storage. The catenation numbers contained within the P2R-Order event allows the reliable-topic repository to determine the order in which these messages were generated and to determine if messages were lost in transit.

6.2.1.3 Repository Processing of Published Message

Upon receipt of a message (issued over one of its managed reliable-topics), the repository queues the message in a temporary buffer, this message is not acted upon until the corresponding P2R-Order event is received. The repository also checks with the identifier contained in the event to see if this event has been previously received at the repository. If a message correlated with this duplicate event is in the repositorys temporary buffer, that message is also discarded as a duplicate.

If the published message has been received, an acknowledgement R2P-ACK event is issued back to the publisher. If the published message corresponding to the P2R-Order event is missing, the repository issues a negative acknowledgement R2P-NAK event to the publisher to retrieve the missing message. The repository also checks to see if there are any gaps in the P2R-Order events received from the publisher. When the repository stores the message it assigns a monotonically increasing *sequence number*.

6.2.1.4 Processing Repository Acknowledgements

A positive acknowledgement R2P-ACK event signifies successful receipt of the message and the corresponding P2R-Order event at the repository. The local buffer entry corresponding to this message can then be removed. Upon receipt of the negative acknowledgement R2P-NAK event the message(s) corresponding to the specified catenation number(s) are retrieved and prepared for retransmission. The retransmission occurs in the P2R-Retransmit event which contains both the original published message along with the catenation number for the message.

6.2.1.5 Message Storage and Persistence Notifications

Upon successful receipt of a published message at the repository, in addition to the operations outlined in section 6.2.1.3 the repository performs three additional functions.

First, depending on the topic type contained in the original published message, the repository loads the appropriate matching engine to compute destinations for the published message based on the registered subscriptions.

Second, the repository adds an entry to the dissemination table that it maintains. For a given sequence number, the dissemination table enables a repository to keep track of destinations that have not explicitly acknowledged the receipt of the corresponding published message.

Finally, the repository issues an event signifying the persistence of the published message. If S is the set of registered subscribers to a given reliable-topic, and if S* is the subset of subscribers whose subscription constraints are satisfied by the published message, then the R2S*-Persistent event signifies that it would be received only by that subset of subscribers.

The R2S*-Persistent event contains the sequence number assigned to the published message and also the identifier associated with the published message. A subscriber

can then correlate a published message and its persistent event.

6.2.1.6 Processing Persistent Events at the Subscriber

A subscriber to a reliable-topic receives published messages from the publishers, and events from the repository. Upon receipt of a message from a publisher, a subscriber stores this message in its temporary local buffer. A subscriber releases a message only if both the message and the corresponding R2S*-Persistent event have been received.

If the subscriber has received both the message and the corresponding R2S*-Persistent event, it proceeds to issue an acknowledgement to the repository. If the subscriber encounters a R2S*-Persistent event without the corresponding published message it concludes that the message was lost in transit. It issues a S2R-NAK event with the missing sequence number(s) for a given reliable-topic to retrieve the corresponding messages.

6.2.1.7 Processing Subscriber Acknowledgements

Upon receipt of an acknowledgement from the subscriber, the repository checks the dissemination table to see if there are any un-acknowledged messages within the range of sequence numbers contained in the S2R-ACK event.

On receipt of the S2R-ACK event from a subscriber, the repository updates the dissemination table entries corresponding to the sequence(s) contained in the event to reflect the fact that the subscriber received messages corresponding to those persistent event sequences.

The repository maintains a sync for every subscriber to the reliable-topics that it manages. The subscriber sync corresponds to the sequence number up until which the repository is sure that this subscriber has received all preceding messages. A subscriber maintains a local copy of this sync.

If the subscriber has received all the messages that it was supposed to receive, and if there were no missed messages between the subscribers current sync and the highest sequence number contained in the S2R-ACK event, the repository advances the sync point associated with this subscriber and issues a R2S-Sync event which notifies the subscriber about this sync advancement. Only upon receipt of this event is the subscriber allowed to advance its sync.

It is possible that the repository, based on the S2R-ACK event, detects that there are some persistent event sequences which were not explicitly acknowledged by the subscriber. The repository assumes that these un-acknowledged messages were lost in transit to the subscriber.

After the detection of missed sequences, the repository issues an R2S-Rectify event, which contains information pertaining to the clients sync advancement (if it is possible) and also the sequencing information and message-identifiers of the missed messages.

6.2.1.8 Processing Errors and Syncs Advances

Upon receipt of the R2S-Rectify event a subscriber performs three steps. First, the subscriber checks to see if any of the messages that it maintains in its temporary buffer has the identifier(s) corresponding to those listed in the R2S-Rectify; this accounts for the case where the R2S*-Persistent event was lost in transit to the subscriber, but the original published message was not. If the message exists in the temporary buffer, the message is delivered.

Second, the subscriber then proceeds to issue a S2R-NAK negative-acknowledgement event to the repository while excluding messages that were reliably delivered in the previous step. The S2R-NAK issued by the subscriber corresponds to the case where messages corresponding to the listed sequence numbers were lost in transit.

Finally, the subscriber advances its sync based on the advancement contained in

the R2S-Rectify event. Note that this is also done in response to the R2C-Sync event.

6.2.1.9 Subscriber and Publisher Recovery

When a subscriber reconnects to the broker network after failures or a prolonged disconnect, it needs to retrieve the missed messages published over a reliable-topic. The recovering entity issues a recovery request S2R-Recovery for every reliable-topic that it had previously subscribed to.

Upon receipt of the recovery request, the repository scans the dissemination table starting at the sync associated with the client. The repository then generates an R2S-Rectify event, which is processed by the subscriber to advance its local sync and also to initiate retransmissions as described earlier in section 6.2.1.8.

In the case of publisher recovery, the repositorys recovery response includes the last known catenation number for a given reliable-topic to which the publisher published.

6.2.2 NaradaBrokering Reliable Delivery Service Extensions

We extend the NaradaBrokering Reliable Delivery Service with a repository replication algorithm[*Bulut*, 2007]. This impacts the reliable delivery protocol and can be summarized as follows:

- A Publisher uses a monotonically increasing catenation number each topic it is publishing to. This is necessary for the repository to detect lost messages. Detecting gaps from a monotonically increasing sequence of catenation numbers is faster than detecting gaps from a sequence of ¡previous catenation number, catenation number¿ pair.

- To reduce the number of events exchanged between repository-publisher and repository-subscriber, we propose a lazy acknowledgment scheme. For this, the repository issues a R2P-NAK event to the publisher every few (configurable)

seconds and the subscriber issues an S2R-ACK event to the repository every few (configurable) seconds. This allows us to include a list of lost message catenations into a single R2P-NAK event and a list of catenation numbers into a single S2R-ACK event, hence reducing the number of R2P-NAK and S2R-ACK events. This scheme can be viewed as lazy negative acknowledgement because repository issues NAK to the publisher including a list of catenation numbers of missed events and lazy positive acknowledgement because the subscriber issues ACK to the repository including a list of catenation numbers of received events.

- Upon the receipt of a P2R-Order event, the repository stores the corresponding message without waiting for missed messages. For lost messages, an entry is added to a storage table with a status flag to indicate that the message is being recovered and needs to be stored. A sequence number is also assigned for those missed events. This procedure avoids accumulation of messages. For example, suppose message i and all previous messages are stored successfully without any gaps between them. The messages from i+2 to i+20 arrive with the corresponding P2R-Order events which carry the catenation numbers of the corresponding messages. The repository assigns sequence numbers to missed events i+1 and messages from i+2 to i+20. An entry for message i+1 is added to the storage table with a flag indicating that it has not been received yet. Then repository stores messages from i+2 to i+20 successfully and publishes R2P-ACK event for them. For missed message i+1, the entry is overwritten as a message arrives. In the previous case, messages from i+2 to i+20 were not stored until message i+1 was received successfully. Waiting for message i+1 to store all messages afterwards also would cause burst in storing messages and publishing R2P-ACK events.

6.2.3 Redundant and Fault-tolerant Repository/Archiving Service

In the previous sections, we outlined our strategy to ensure a reliable delivery. In this scheme if there is a failure at the repository(i.e. Archiving Service), the clients interested in reliable communication, over any of the managed reliable-topic, need to wait for this repository to recover prior to the reliable delivery guarantees being met. We now extend this scheme to ensure that the reliable delivery guarantees are satisfied in the presence of repository failures. To achieve this, we include support for multiple repositories constituting a repository-bundle for a given reliable-topic; it is not necessary that the topics managed by these repositories be identical. A repository may thus be a part of multiple repository-bundles at the same time.

We support a flexible redundancy scheme with easy addition and removal of repositories that manage a given reliable-topic. There are no limits on the number of repositories for a given reliable-topic. This scheme can sustain the loss of multiple repositories: in a system with N repositories for a given reliable-topic N-1 of these repositories can fail, and reliable delivery guarantees are met so long as at least one repository is available.

The repositories that constitute the repository-bundle for a given reliable-topic function autonomously. At any given time, for a given reliable-topic, a client communicates with exactly one repository within the corresponding repository-bundle. This entity is also allowed to replace this repository with any other repository within the bundle at anytime.

Besides additional redundancy, and the accompanying fault-tolerance, a highly-available, distributed repository scheme enables clients to exploit geographical and network proximities. Using repositories that are closer ensures reduced latencies in the receipt of events from the repository. Packet loss-rates typically increase with the number of intermediate hops (for UDP and Multicast).

Besides the selection of the repository from a repository-bundle, as part of the

bootstrap, operations at the clients are identical to those in place for a single repository.

6.2.3.1 Steering Repository

A publisher or a subscriber to a reliable-topic can interact with exactly one repository within the repository-bundle for that reliable-topic; this repository is referred to as the steering repository for that publisher/subscriber. At any time, a client is allowed to replace its steering repository with any other repository from the repository bundle.

Every repository within the bundle keeps track of a clients delivery sequences passively and actively. For a given entity, at any given time, there will be one steering repository operating in the active mode by initiating error-corrections and retransmissions. Other repositories operating in the passive mode do not initiate these actions.

At every repository, within the repository-bundle for a given reliable-topic, the list of registered clients is divided into two sets - those that the repository steers and those that it does not. The repository operates in the active mode for the steered clients and in the passive mode for the clients that it does not steer. In the active mode, a repository performs all functions outlined in section 6.2.1. In the passive mode, a repository listens to all events initiated by the publishers and subscriber; however, the repository will not issue events related to reliable communications to the clients that it does not steer. Operating in the passive mode, allows a repository to take over as the steering repository for clients that it does not presently steer.

When a client is ready to initiate reliable communications, it has to designate a steering repository from the set of repositories within the repository-bundle associated with the reliable-topic. Selection of the steering repository is done based on network proximity using probes to compute network round-trip delays to the repositories. The client then issues an event over the repositorys communications-topic designating it

as the steering repository. Upon receipt of this event, the repository adds that client to its list of steered clients.

6.2.3.2 Ordered Storage of Published Messages

For every published message, the publisher issues a P2R-Order event (where R is the repository-bundle), which is received by all repositories within the repository-bundle. This allows all repositories within the repository-bundle to keep track of the published messages. However, only the steering repository (operating in active mode) for this publisher is allowed to issue the R2P-ACK and R2P-NAK events to acknowledge the receipt of messages and to initiate retransmissions respectively.

Retransmissions issued in response to the R2P-NAK event are sent to all repositories using the P2R-Retransmit event. The rationale for this is that if a message was lost in transit to the publishers steering-repository, there is a good chance that the message (or the corresponding P2R-Order) event was also lost in transit to the other repositories.

6.2.3.3 Generation of Persistence Notification

Once a published message is ready for storage at the repository, the message is assigned a sequence number and is stored onto stable storage along with the published message. In this scheme each repository is autonomous, and thus maintains its own sequencing information. This implies that a message published by a publisher, may have different sequence numbers at different repositories. It follows naturally that the sync associated with a given subscriber can be different at different repositories. However, the catenation number associated with a publisher is identical at every repository within the repository-bundle.

A repository computes destinations associated with every published message. These destinations are computed based on the subscriptions registered by subscribers

to this reliable-topic irrespective of whether subscribers they are steered by the repository or not. The repository then proceeds to issue a persistence notification. The topic associated with the R2S*-Persistent event is such that it is routed only to the subset S* of its steered subscribers with subscriptions that are satisfied by the topic contained in the original message.

6.2.3.4 Acknowledgements, Errors and Syncs

Upon receipt of R2S*-Persistent events from its steering repository, a subscriber proceeds to issue acknowledgements. This acknowledgement, the S2R-ACK is issued over the repository-bundle communications topic. Since, the message is received by the repository-bundle, all repositories are aware of delivery sequences at different subscribers. The S2R-ACK event contains sequence numbers corresponding to its steering repository and also includes the identifier associated with the steering repository.

Error correction, and sync advancements, for a given subscriber is initiated by its steering repository through the R2S-Rectify event. Retransmission requests by a subscriber are targeted by its steering repository in the S2R-NAK event.

6.2.3.5 Gossips between Repositories

Repositories within a repository-bundle gossip with each other. Repositories within a repository-bundle need to exchange messages about the registration/deregistration of clients to a managed reliable-topic as well addition and removal of subscriptions by clients to a reliable-topic. A given repository stores each of these actions and assigns each action the next available sequence number.

6.2.3.6 Dealing with Repository Failures

A publisher detects a failure in its steering repository, if it does not receive R2P-ACK events for published messages within a certain time duration. A subscriber detects a steering-repository failure if it receives published messages to reliable-topics, but no corresponding persistence notifications from its steering repository. These clients then proceed to discover a new steering repository. The publisher then exchanges information about its catenation number with the replacement steering repository. If there is a mismatch wherein the steering repository's catenation is lower than that at the publisher, the repository proceeds to retrieve this message from a repository within the bundle.

6.2.3.7 Recovery of a Repository

Upon recovery from a failure, it needs to discover an assisting-repository: this is a repository within the repository bundle that is willing to assist the repository in the recovery process. The recovering replica first checks to see if the list of registered clients and subscriptions have changed, and proceeds to retrieve updates to this list.

Next, the repository proceeds to retrieve the list of catenation numbers associated with the publishers. Based on these catenation numbers, the repository computes the number of missed messages and proceeds to set aside the corresponding number of sequences. For messages (missed and real-time) that it stores, a recovering-repository issues Gossip-ACK acknowledgements at regular intervals.

The recovering-repository proceeds to do two things in parallel. First, it proceeds to retrieve missed messages from the assisting repository. For every missed message the recovering-repository also retrieves a dissemination list associated with it. This allows a repository to keep track of the subscribers that have not acknowledged these messages. Additionally, the repository-table entries corresponding to each message are also retrieved. A repository cannot be the steering repository for any entity till

all the missed messages have been retrieved.

Second, it subscribes to various communications topics so that it can start receiving messages published in real-time. The first time a repository receives a message from a publisher, it checks to see if the catenation number associated with that message indicates missed messages. This could happen because the missed message(s) would have been in transit to the assisting repository. Thus, during recovery if the assisting repository reported a catenation number of 2000, and if the catenation number associated with the first real-time message received from the publisher is 2010 it implies that there are 9 additional messages from this publisher that are missed. The repository sets aside 9 sequence numbers, and issues a request to retrieve these messages. The repository also proceeds to store the published message based on the newly advanced sequence number.

6.2.3.8 Addition of a Repository

When a repository is added to the repository-bundle associated with a reliable-topic, the newly added repository takes the following steps. First, it needs to discover an assisting-repository: this is a repository which is present in the repository bundle and one which is willing to assist the repository in the addition process.

Second, the repository retrieves the list of registered clients, and the subscriptions registered by the registered subscribers. As described in section 6.2.3.7 the repository then proceeds to retrieve missed messages along with the corresponding dissemination lists and repository-table entries in addition to processing real-time messages.

6.2.3.9 Graceful Removal of a Repository

When a repository is ready to leave a repository bundle, it proceeds to issue an event to its active steered clients, requesting them to migrate to another repository. The departing-repository then operates in silent mode as far as the clients

are concerned. The departing-repository also gossips with other repositories within the repository-bundle to check if the catenation numbers associated with previously steered publishers is greater than or equal to its last known value at the departing-repository.

Once a repository has confirmed that all messages published by its previously steered publisher have been received at one of the repositories within the bundle, it is ready to leave the repository-bundle. The departing repository then simply issues a Gossip-Leave event. Repository table entries corresponding to this repository will no longer be maintained at other repositories.

CHAPTER VII

Experiments of Scalability and Robustness

7.1 Performance Experiments on Desktops

7.1.1 Resource usage Test

To prove the scalability and robustness of our system, we did several performance tests on the stream archiver by feeding different numbers of multimedia streams in different formats at the same time. CPU usages of the running archiver process are logged and displayed in the following Figure 7.1.

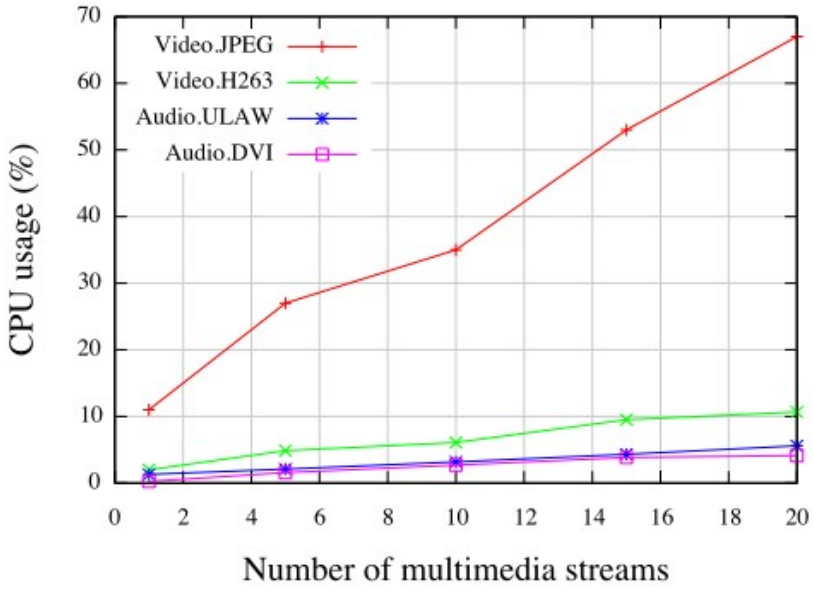

Figure 7.1: CPU Usages of A Stream Archiver Saving Multimedia Streams

The experiments were done on an Intel Pentium 4 machine with a 3.40GHz CPU and system memory of 1.75G. The results show us that the stream archiver works

pretty well on streams that are made up of events with small payloads, such as audio streams and highly compressed video stream in the figure. Less than 10% CPU was used to process 20 simultaneous Video.H.263 streams. Since a large event payload requires more copy instructions and system I/Os, it is not hard to explain why CPU usages were so high when the stream archiver tried to archive those Video.JPEG streams. We also notice that the CPU usages of brokers in the NaradaBrokering system were also at a quite high level when they are transmitting Video.JPEG streams.

7.1.2 Annotation Latency Test

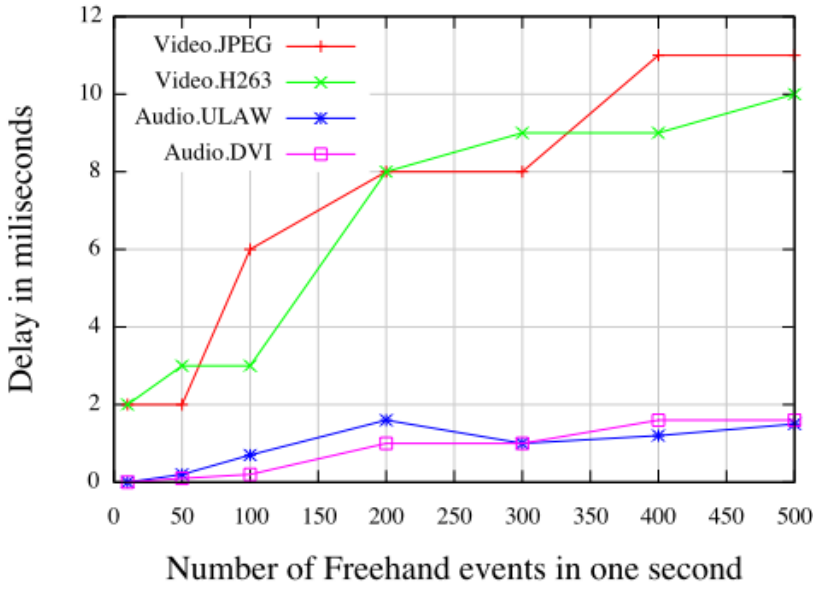

Figure 7.2: Time Delays of Freehand Whiteboard Events

Our desktop client has a built-in whiteboard (see Figure 4.1) to support free-hand drawing annotation as eSports[*Zhai et al.*, 2005] does. It is important that drawings such as lines, shapes and inserted images are displayed timely on remote clients, especially when users are working on real time data streams. Delayed or disordered annotations will cause problems to the real time communication. We tested our system by sending large amounts of free-hand whiteboard events in one second while system users are playing different types of multimedia streams. We record the time

difference between each event's creation time and rendering time at remote clients. The Average of all differences recorded in the same test is used as the final result.

Though ascending, time delays caused by the system are still much lower than the required perception level of delay (200-400ms for video streams) in a cooperation system[*Huang et al.*, 1999]. Distributed users will not have any problems on white-board annotations in the system while they are cooperating on supported real time data streams.

7.2 Performance Experiments on Mobile devices

7.2.1 Resource usage Test

We also conducted two performance experiments on the new mobile extension of the collaborative annotation framework. The first test was to see the resource usages of a typical annotation on video streams sent with different encoding parameters from the desktop client. The mobile client was running on a HTC Inspire 4g android smart phone with 1GHz Scorpion CPU and 768MB internal memory. Multiple video streams were sent to the smart phone with different FPS (frame per second) and quality. The test results are shown in the following Figure 7.2.1.

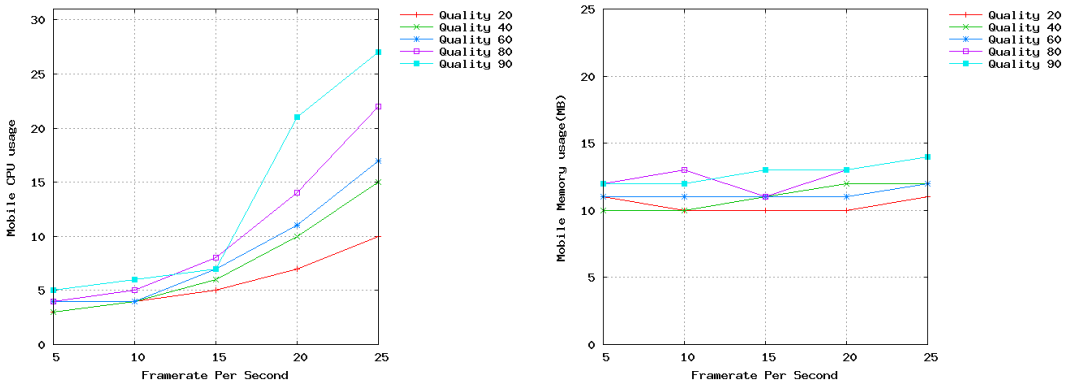

Figure 7.3: Resource usages of playing video streams with different parameters

The test results show that the CPU usage of the smartphone falls below 30% for

a typical H.263 video stream with 25 FPS and video quality 80. And the memory consumption is managed within the limitation of regular android apps (16MB per app). This proves that our mobile client can be quite responsive to user interactions such as adding annotations and replaying past ones.

7.2.2 Latency Test

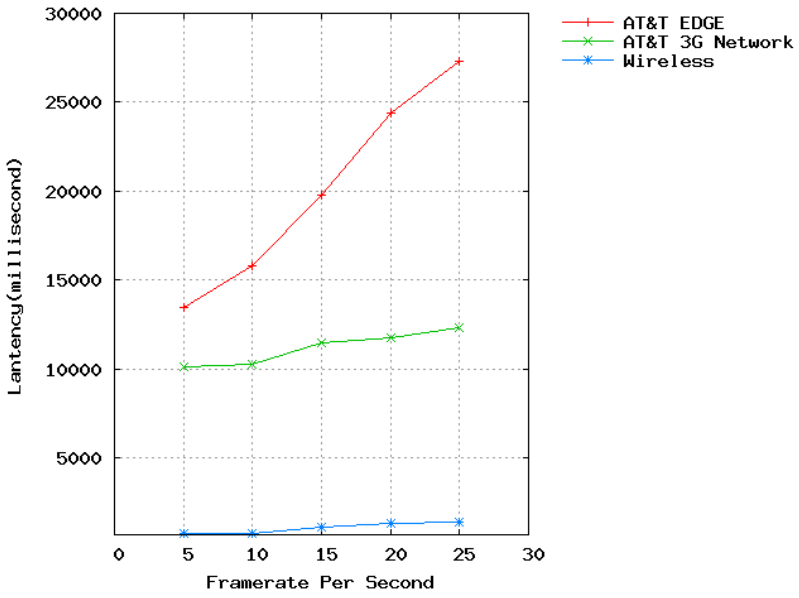

Figure 7.4: Start latency of playing video streams with different parameters on different networks.

The second experiment was designed to evaluate the latency caused by local buffering the streaming data and communication between the proxy and media player. The mobile client was running on three different types of mobile networks: AT&Ts EDGE, 3G and a wireless network. The time was measured between the time that a NaradaBrokering event was received from the desktop client and the time that the android media player starts playing. Figure 7.4 above contains test results of our mobile client receiving video streams with different parameters. It shows that the delay was managed under a reasonable level for our mobile client on wireless networks and we saw expected long delays (longer than 10 seconds) on low bandwidth and unstable

networks such as EDGE and 3G. We however noticed that the latency may be slightly improved by using events with smaller payload size which may speed up the transmission between the broker and the mobile client on those networks. However the improvement was quite limited and we can hardly get a fluent collaboration between the desktop and mobile applications.

7.3 Framework Scalability Experiments

In order to investigate the scalability of event-based infrastructure and our framework implementation, two experiments were conducted to answer the following research questions:

- How is the performance of the archiving and replaying services when the number of such requests increases?
- How fast does the framework respond to possible mobile client loss under heavy payload? This is very crucial to the better user experience of the annotation framework.

7.3.1 Resource usage Test

To answer the first question, we increased the number of simultaneous requests and measured the responding time of the archiving and replaying service. From the results in the Figure 7.5 below, we conclude that our design of the archiving and replaying service can provide prompt response to the end user even if there are hundreds of them using the framework at the same time. The logged responding time increased a lot after a particular threshold(400) is root caused by the hardware limitation of the server these tests were conducted on. The disk I/O was logged as 100% (using iotop on the linux server) during those scenarios.

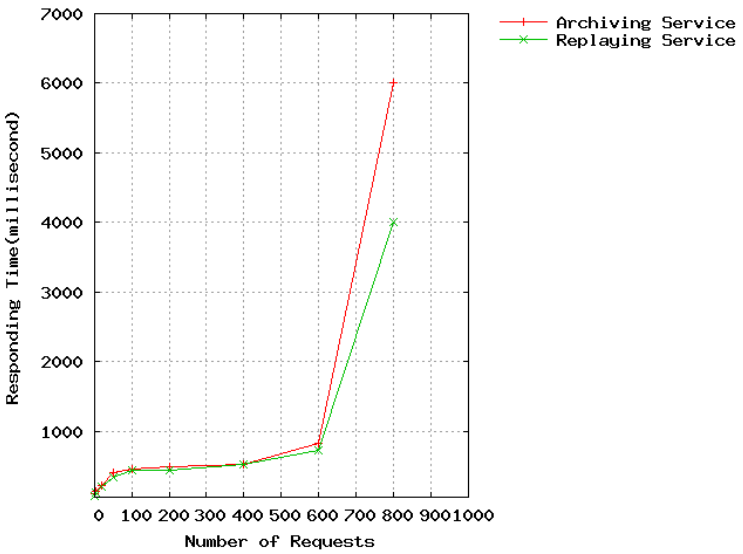

Figure 7.5: Responding time of Archiving & Replaying Service for different number of requests.

7.3.2 Latency Test

The second question was answered by monitoring the time before the session list changes in the session server after manually disconnecting several mobile clients(by killing the application). We increased the number of events being transmitted over the event-based network which simulates different level of system loads on the framework.

Messages per Second	**Average time(ms)**
0	0
314	0.73
876	1.11
1219	1.79
1682	2.41
2013	2.99
2508	3.87

Table 7.1: Average Time Before Session List Changes under Different System Loads

Table 7.1 above shows us that the annotation framework is quite responsive to session changes even under reasonable system loads. The average time before the session list updates is controlled at a reasonable level(less than 3 seconds) which minimizes the user experience impact caused by possible client disconnections.

CHAPTER VIII

Conclusions and Future Work

8.1 Summary

In this thesis, we introduce a framework system that supports collaborative annotation on generic data streams. It enables sending, browsing, rendering and annotation on real time data streams in distributed environments and our experiment results show that it works properly for compressed data streams under high stress circumstances.

We also present our efforts to extend the collaborative annotation framework into the mobile environment. We have implemented a user friendly prototype of the mobile client with event translating proxies for the mobile users to collaborate with desktop users easily. The performance experiments show that our design can provide satisfying user experience on android-based mobile devices with wireless connection.

8.2 Conclusion

This framework expands its scope of application through generalizing the procedure of data stream processing and defining basic stream capturing and rendering interfaces. Users are able to quickly extend the system by writing their own stream sources/renders. Through implementing those interface methods, we can support more types of data streams other than mere multimedia ones in the system, which makes it more capable of satisfying diverse application requirements. The system also

provides a simple user interface to simplify the manipulation of streaming data and it also supports annotation on live data streams via local stream buffers.

8.3 Future work

Our next step is to continue the development of this prototype to improve its stability. More stream sources and renders will be added to the system to support data streams generated by non-multimedia sources such as earthquake sensors, hand-held devices and medical instruments. A configuration detector will be added to the system to simplify the recognition of new StreamSource and StreamSink. We plan to standardize our annotation metadata format into MPEG-7 compatible version so that we can have more accurate search functionality. To improve the stability and performance of the system on low bandwidth networks and conduct further experiments for more sophisticated use cases. We also plan to apply the same design on other mobile platforms such as iOS and Windows mobile.

8.4 List of Publications Related to This Thesis

Following is a list of publications directly related to this thesis:

- Tao Huang, Geoffrey Fox *"Collaborative Annotation of Real Time Streams on Android-Enabled Devices"* Workshop 13-IoT Internet of Things, Machine to Machine and Smart Services Applications (IoT 2012) at The 2012 International Conference on Collaboration Technologies and Systems (CTS 2012) May 21-25, 2012 The Westin Westminster Hotel Denver, Colorado, USA, Technical Report February 14 2012

- Tao Huang, Shrideep Pallickara, Geoffrey Fox *"A Distributed Framework for Collaborative Annotation of Streams"* Proceeedings of The 2009 International

Symposium on Collaborative Technologies and Systems CTS 2009 May 18-22, 2009 The Westin Baltimore Washington International Airport Hotel Baltimore, Maryland, USA

- Wenjun Wu, Tao Huang, Geoffrey Fox *"Building Scalable and High Efficient Java Multimedia Collaboration"* Proceedings of IEEE 2006 International Symposium on Collaborative Technologies and Systems CTS 2006 conference Las Vegas May 14-17 2006; IEEE Computer Society, Ed: Smari, Waleed & McQuay, William, pp18-25. ISBN 0-9785699-0-3 DOI

- Wenjun Wu, Geoffrey Fox, Hasan Bulut, Ahmet Uyar, Tao Huang *"Service Oriented Architecture for VoIP conferencing"* Special Issue on Voice over IP - Theory and Practice of the International Journal of Communication Systems Volume 19, Issue 4 , Pages 445 - 461 Edited by John Fox, P. GburzynskiDOI

REFERENCES

REFERENCES

3GPP (2010), 3GPP Long Term Evolution Standard, https://sites.google.com/site/lteencyclopedia/home.

Amazon LLC (2006), Amazon simple storage service (amazon s3), http://aws.amazon.com/s3/, [Online; accessed 30-September-2012].

amqp (2012), AQMP Open standard for messaging middleware, http://www.amqp.org/confluence/display/AMQP/Advanced+Message+Queuing+Protocol.

Anguera, X., J. Xu, and N. Oliver (2008), Multimodal photo annotation and retrieval on a mobile phone, in *Proceedings of Multimedia Information Retrieval*, pp. 188–194.

Apache (2007), Apache ActiveMQ open source messaging system, http://activemq.apache.org/.

Bargeron, D., A. Gupta, J. Grudin, Sanocki, E., Li, and F (2001), Asynchronous collaboration around multimedia and its application to on-demand training, in *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34)*.

Berez, A. L. (2007), Review of eudico linguistic annotator (elan), *Language Documentation & Conservation 1(2)*, pp. 283–9.

Bertenthal, B., et al. (2007), Social informatics data grid, in *e-Social Science Conference*.

Bulut, H. (2007), High performance recording and manipulation of distributed streams, Ph.D. thesis, Indiana University Bloomington.

Childers, L., T. Disz, R. Olson, M. E. Papka, R. Stevens, and T. Udeshi (2000), Access grid: Immersive group-to-group collaborative visualization.

Eclipse (2012), SWT Library, http://www.eclipse.org/swt/.

Edwards, J. (Ed.) (1999), *3-Tier Server/Client at Work, 1st edition*, John Wiley & Sons.

El-Saban, M., X.-J. Wang, N. Hasan, M. Bassiouny, and M. refaat (2011), Seamless annotation and enrichment of mobile captured video streams in real-time, in *IEEE Internation Conference on Multimedia and Expo (ICME)*.

Google Inc (2005), Google Docs, http://docs.google.com.

Google Inc (2009), Open handset alliance, http://www.openhandsetalliance.com/.

Google Inc (2010), Google goggle, http://www.google.com/mobile/goggles.

Google Inc. (2012), Google drive, https://www.google.com/intl/en/drive/start/index.html, [Online; accessed 30-April-2012].

Huang, L., M. Iijima, and K. Sezaki (1999), A survey on human perception of delay in a cooperation system, in *IEICE Communications Society Conference*.

Huang, T., S. Pallickara, and G. Fox (2009), A distributed framework for collaborative annotation of streams, in *Proceedings of 2009 International Symposium on Collaborative Technologies and Systems*, pp. 440–447.

IBM (2002), WebSphereMQ, http://www-01.ibm.com/software/integration/wmq/.

IBM Corp. (2012), Ibm big data, http://www-01.ibm.com/software/data/bigdata/, [Online; accessed 30-Jan-2012].

ISO/IEC Moving Picture Experts Group (2004), Mpeg-7 standard, http://mpeg.chiariglione.org/standards/mpeg-7/mpeg-7.htm.

ITU Recommendation G.711 (1988), Pulse Code Modulation (PCM) of Voice Frequencies.

ITU Recommendation H.225 (2000), Calling Signaling Protocols and Media Stream Packetization for Packet-based Multimedia Communication Systems.

ITU Recommendation H.243 (1998), Terminal for low bit-rate multimedia communication.

ITU Recommendation H.245 (2000), Control Protocols for Multimedia Communication.

ITU Recommendation H.261 (1991), Video Codec for Audiovisual Services at p x 64 kbit/s.

ITU Recommendation H.263 (1998), Video coding for low bit rate communication.

ITU Recommendation T.120 (1996), Data Protocols for Multimedia Conferencing.

IVCi LLC (2009), Tandberg VCS, http://www.ivci.com/videoconferencing-tandberg-video-communication-server.html.

Karim, A. (2000), H.323 and associated protocols, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.8534.

Koskelainen, P., H. Schulzrinne, and X. Wu (2002), A sip-based conference control framework, in *ACM Press New York*.

Kratz, M., M. Ackerman, T. Hanss, and S. Corbato (2001), NGI and Internet2: Accelerating the Creation of Tomorrow's Internet, *Stud Health Technol Inform*, pp. 84(Pt 1):28–32.

Layar (2012), Layar Vision, http://www.layar.com/.

Microsoft (2010), Windows azure, http://www.windowsazure.com/, [Online; accessed 30-September-2012].

Microsoft Corporation (2011), Office 365, www.microsoft.com/Office365.

Mule Soft (2010a), Mule ESB, http://www.mulesoft.com/.

Mule Soft (2010b), Mule MQ open source enterprise-class Java Message Service (JMS) implementation, http://www.mulesoft.org/documentation/display/MQ/Home.

OpalVoip, and H323Plus (2007), Office 365, http://openh323.sourceforge.net.

Open Handset Alliance (2009), Android Multimedia Framework, http://developer.android.com/guide/topics/media/index.html.

Oracle (2001), Java Message Service JMS API, http://www.oracle.com/technetwork/java/index-jsp-142945.html.

Oracle (2004), Oracle Java Media Framework API, http://www.oracle.com/technetwork/java/javase/download-142937.html.

Pallickara, S., and G. Fox (2003), Naradabrokering: A distributed middleware framework and architecture for enabling durable peer-to-peer grids, in *Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003, Rio Janeiro*, pp. 41–61.

Polycom Inc (2000), VideoStation MP Series, http://www.it-telecoms.org/video-conferencing-h323.html.

Ren, W., and S. Singh (2005), An automated video annotation system, in *ICAPR (2)*, *Lecture Notes in Computer Science*, vol. 3687, edited by S. Singh, M. Singh, C. Apté, and P. Perner, pp. 693–700, Springer.

Savakis, A., P. Sniatala, and R. Rudnicki (2003), Real-time annotation using mpeg-7 motion activity descriptors, in *Mixed Design of Integrated Circuits and Systems 10*.

Schroeter, R., J. Hunter, and D. Kosovic (2003), Vannotea - a collaborative video indexing, annotation and discussion system for broadband networks, in *In K-CAP 2003 Workshop on Knowledge Markup and Semantic Annotation*.

Shevade, B., H. Sundaram, and M. Yen-Kan (2005), A collaborative annotation framework, in *2005 IEEE International Conference on Multimedia and Expo*, pp. 1346–1349.

Smith, J. R., and B. Lugeon (2000), A visual annotation tool for multimedia content description, in *Proc. SPIE Photonics East, Internet Multimedia Management Systems*.

Stein, L. (2001), Genome annotation: from sequence to biology, *Nature Reviews Genetics*, *2*, 493–503 (July 2001), doi:10.1038/35080529.

VMWare (2010), RabbitMQ open source Enterprise Messaging System, http://www.rabbitmq.com/.

W3C RDF Group (2004), RDF specification, http://www.w3.org/RDF/.

Wang, J., and J. Canny (2006), End-user place annotation on mobile devices: A comparative study, in *CHI*.

Wang, Z., O. Gnawali, K. Heath, and L. Guibas (2011), Collaborative image annotation using image webs, in *Proceedings of the Army Science Conference (ASC 2010)*.

Want, R. (2011), Near field communication, *IEEE Pervasive Computing*, *10*(3), 4–7.

Wilhelm, A., Y. Takhteyev, R. Sarvas, N. House, and M. Davis (2004), Photo annotation on a camera phone, in *Proc. CHI Extended Abstracts*, pp. 1403–1406.

Wu, W., T. Huang, A. F. Mustacoglu, M. Pierce, and G. Fox (2006), Globalmmcs collaborative clients and services for portals.

Wu, X., P. Koskelainen, H. Schulzrinne, and C. Chen (2002), Use sip and soap for conference floor control, in *Internet Engineering Task Force*.

Yeh, T., K. Tollmar, and T. Darrell (2004), Searching the web with mobile images for location recognition, in *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition (CVPR 04)*, p. 7681.

YouTube LLC (2012), Youtube statistics, http://www.youtube.com/t/press_statistics, [Online; accessed 21-October-2012].

Zhai, G., G. Fox, M. Pierce, W. Wu, and H. Bulut (2005), esports: Collaborative and synchronous video annotation system in grid computing environment, in *Proceedings of IEEE International Symposium on Multimedia (ISM2005)*.

APPENDICES

APPENDIX A

Vitae

Name of Author:	Tao Huang
Place of Birth:	Jingzhou, People's Republic of China
Degrees Awarded:	
Jan 2013	Ph.D. in Computer Science
	Indiana University
	Bloomington, IN, U.S.A.
March 2004	M.S. in Computer Science
	Beihang University
	Beijing, China
July 2001	B.S. in Computer Science & Engineering
	Beihang University
	Beijing, China

Experience:

May 2012 Jan 2013	Software Development Engineer Microsoft Corporation Redmond, WA, U.S.A.
Jan 2005 April 2012	Graduate Research Assistant Pervasive Techology Institute Indiana University Bloomington, IN, U.S.A.
Sept 2001 Feburary 2004	Graduate Research Assistant School of Computer Science Beihang University Beijing, China