ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 4, June 2013

Towards a Distributed Quality of Service Handler for Multimedia Presentations

Loay S. Ismail

Assistant Professor, Dept. of Computer Science and Engineering, College of Engineering, Qatar University, Doha, Qatar

ABSTRACT:Distributed multimedia presentation, involving different media objects on different servers, must strictly follow its defined temporal scenario. Out of synchronization is a fundamental problem in case of distributed multimedia. In this paper, we introduce a distributed QoS handler, whose aim is to distribute the QoS adaptation tasks among all the servers holding the media object. In this case, the client is considerably freed from the QoS-related tasks and it focuses only on playing the received, adapted media objects and notifying the end of their presentation to the concerned servers. This will lead to a better load balance for playing a distributed multimedia application, where the tasks of applying QoS rules and playing the objects are better distributed among the playing client and the media object servers.

Keywords: Quality of service, multimedia presentation, distributed system, synchronization.

I. INTRODUCTION

A multimedia presentation is the process of playing a set of media objects, such as video, audio, still images, and text, according to a temporal scenario, previously specified by a multimedia application author. These different media objects may reside on different sites on the network. In this case, the presentation is classified to be a distributed multimedia presentation. The distribution of the media objects over the network imposes the risk of entering in an *out-of-synchronization* state due to the random values of end-to-end delays and networks congestion in the paths between the client that runs the presentation and the different servers that hold the media objects.

Thus, a mechanism of Quality-of-Service (QoS) must be put in place at the application level, to guarantee a minimum level of presentation quality. In most of the time, a multimedia presentation can restore its synchronization through some acceptable degradation in the presentation quality of media objects [1]. The degree of quality degradation must be agreed upon between the client and the server. The way of degrading the presentation quality of media objects differs from one media type to another. For example, in the case of video objects, we can skip frames, reduce number of colors, or reduce size of frames, but in case of audio objects, we can use broadcast or even telephone quality instead of hi-fi quality for audio objects.

Most of the current work on QoS support for multimedia traffic acknowledges the need to support application-level QoS. In response, many network QoS control schemes with resource reservation planning have been designed [1]. Generally, these QoS control schemes provide guaranteed QoS to users by reserving resources based on the worst-case analysis, and this reservation is sustained for the entire transmission period. Although QoS is guaranteed, it leads to low resource utilization. This poses a challenge to design suitable QoS control schemes, which should help in handling multimedia traffic with its desired QoS characteristics, and with optimum resource utilization.

In this study, we propose a mechanism for an application level distributed QoS handler that allows servers to act in an autonomous way in taking QoS decisions and performing QoS actions, while sending the media objects data to the client. In section II, we give an overview of the proposed system. In section III, we present how to generate plan of actions for different servers starting from a temporal scenario. Then in section IV, we present how the plans of actions are concurrently executed. Then finally, we conclude the paper in section V.

II. SYSTEM OVERVIEW

In this section, we will present an overview of the suggested QoS handler.The temporal scenario of the multimedia presentation is written using the Synchronized Multimedia Integration Language (SMIL) [2], which is a declarative language that declares the media objects taking part in the presentation and how they are related in time and space. When a SMIL document is chosen to be played, the document will be passed to the scenario parser to parse the document and convert the declared scenario from the textual format to an intermediate representation that can be analyzed and handled by the scheduler. This intermediate representation of the temporal scenario can be in one of the following forms: hierarchical directed acyclic graphs [3], timelines [4], temporal Petri-nets [5], or any other form of representation that can hold data about media objects, and the temporal and spatial relationships between them.

ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 4, June 2013

Now, the intermediate representation of the scenario is ready to be analyzed by the scheduler at the client side. The result of the analysis is composed of the list of the media objects taking part in the representation and their exact locations on the network, i.e. addresses of servers holding the media objects. Also, the scheduler will extract the *plan-of-actions* (PoA) for the client as well as for each of the servers in the list. A *plan-of-actions* (PoA) is a sequence of scheduled actions needed to be performed by the server on every media object residing on it, as an initial phase of playing the object. The scheduling of these actions is based on the temporal scenario to be presented. Scheduled actions can be one of the following:

1. applying a QoS rule on media data,
2. adapting media data according to a given QoS rule, or
3. sending the adapted media data to an appropriate media player running on the client.

Every PoA is to be sent to its corresponding server to be executed locally at each server. The PoAs are generated in a way that allows the different PoAs to communicate and exchange status information about the progress of their servers. This communication facility allows for the synchronization between the different servers to comply with the original scenario. The PoA is responsible of applying QoS rules, if needed, before sending the media data to the client.

The servers themselves may be using different platforms. In other words, the servers may be running on different types of hardware and system software. So, the PoA must be platform independent to allow it to run on any hardware-software configuration.

In the next section, we will present the mechanism of generating *plans-of-actions* (PoAs) used by the distributed QoS handling system.

III. Generation of Plans of Actions

The PoAs generated by the scheduler must be in a standard, exchangeable, human readable format. So, we chose the XML [6] syntax to describe the PoAs. The XML format of the PoA can be used as a generic pivot format for converting this multimedia document to any other target format, which can be either a programming language or any other non-executable format.

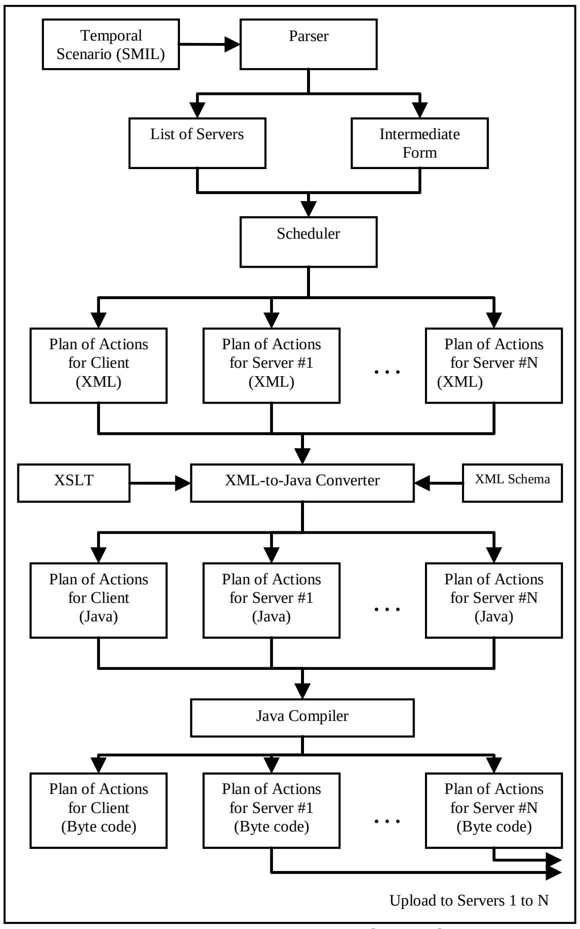
Temporal Scenario (SMIL)
Parser
List of Servers
Intermediate Form
Scheduler
Plan of Actions for Client (XML)
Plan of Actions for Server #1 (XML)
. . .
Plan of Actions for Server #N (XML)
XSLT
XML-to-Java Converter
XML Schema
Plan of Actions for Client (Java)
Plan of Actions for Server #1 (Java)
. . .
Plan of Actions for Server #N (Java)
Java Compiler
Plan of Actions for Client (Byte code)
Plan of Actions for Server #1 (Byte code)
. . .
Plan of Actions for Server #N (Byte code)
Upload to Servers 1 to N

Fig. 1 System Overview at Client Side

In our system, see Fig. 1, we chose the Java language as the programming language for its portability and independence of platforms. So, an XML-to-Java converter can be put in place to make the conversion from XML to Java format using Document Type Definition (DTD) [7],XML Schema[8]or REgularLAnguage for XML Next Generation (RELAX NG) [9] to represent the structure of the generated Java document, and Extensible StylesheetLanguage Transformations (XSLT) [10] to represent the layout of the Java document.

A separate Java document will be automatically generated for each PoA, i.e. one Java document for each server taking part in the multimedia presentation. Then, all of the generated Java documents will be compiled producing a corresponding byte-code (class) file. Each generated byte-code file will be uploaded to its corresponding server. The byte-code file is a platform independent, executable code that can be executed on any software/hardware platform, provided that the platforms, under consideration, have appropriate Java Virtual Machines (JVM) installed on them. Thus, each server will be able to execute its PoA using the JVM installed on it.

As previously mentioned, the PoAs are generated in a way allowing the different servers to communicate between each other for synchronization purposes. So, the client-servers communications, needed for synchronization, can be significantly reduced, leading to a distributed, decentralized method of inter-servers synchronization. However, the servers and the client are in continuous communication with each other to exchange information about the progress of playing the document on the client side. Moreover, the client can send time access control messages to the servers, such as *start*, *stop*, *pause*, and *resume*, to control the media stream and, consequently, the progress of the presentation.

On each server, the PoA will call a QoS module, which should:

- Probe the network for end-to-end delay and throughput,
- Apply appropriate QoS rules to adapt the media object data according to the current status of the network, the type of media object and the temporal constraint of the object,
- Finally, send the adapted media object data to the appropriate media player on client side.

When the playing of an object ends, a *completion acknowledgement* signal will be sent to the server that stores the media object, and the action plan will respond accordingly.

Consider the simple scenario shown in Fig. 2, written using SMIL temporal operators *SEQ* and *PAR*, for sequential and parallel temporal relations, respectively. The scenario temporally relates four objects *A*, *B*, *C* and *D*, residing on three different servers, *SERVER_1*, *SERVER_2* and *SERVER_3*. Objects *A* and *D* reside on *SERVER_1*, while object *B* resides on *SERVER_2*, and, finally, object *C* resides on *SERVER_3*.

```
<SEQ>
<SEQ>
<PAR>
         http://SERVER_1/A
         http://SERVER_2/B
</PAR>
      http://SERVER_3/C
</SEQ>
http://SERVER_1/D
</SEQ>
```

Fig. 2 Scenario in SMIL

We use the Directed Acyclic Graph (DAG) as our intermediate form. A directed graph is an ordered pair D = (V, E), where V is the set of nodes or vertices and E is the set of directed edges. Each edge is represented as e = (s, d, O), where s is the source node, d is the destination node and O is an attribute of the media object represented by this edge. The attribute O is an ordered pair O = (t, r), where t represents the properties of the media object and r is the server address on which the object resides. Let R = {r} be the list of servers holding the media objects composing the multimedia application.

The PoA of each server is generated as a function of the objects residing at that server and its temporal position with respect to the other objects among all the servers. The synchronization algorithm, used for generating the PoAs of servers R is as follows.

```
$\forall r \in R, W = \{ e \mid e \in E \wedge server(e) = r \}$
$\forall w \in W$
    let $x = n \mid n \in N \wedge n = source(w)$
    let $Y = \{ e \mid e \in E \wedge destination(e) = x \}$
$\forall y \in Y$
        wait ( end ( y ) )
    start ( w )
    wait ( end ( w ) )
    notify ( end ( w ) )
```

Consider Fig. 3 that shows the DAG intermediate form representation for the scenario under consideration (see Fig. 2). The horizontal arrows are the directed edges representing the temporal duration for playing an object, while the vertical lines are the nodes representing the synchronization instants, such that the outgoing edges will start their presentations *only* if the presentations of all of the incoming edges come to an end. For example, objects A and B should start simultaneously with the beginning of the multimedia application presentation, while object C should start playing only when both objects A and B come to an end. Also, object D should start once object C comes to an end.

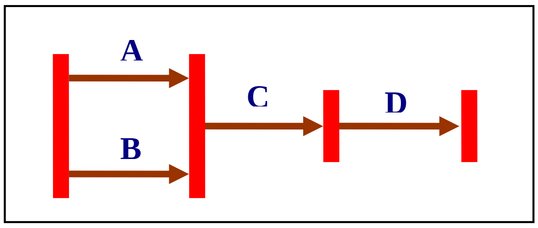

Fig. 3 Intermediate Form of Scenario

IV. Execution of the Plans of Actions

The pseudo-code, of the generated PoA for the client, is shown in Fig. 4, while the PoAs created for the three servers are shown in Figs. 5, 6 and 7. Each server will create a thread to handle its generated PoA. Each media object on the server will be handled by a separate thread, that can serve the delivery of the adapted media to the client. Separate thread per object is essential, especially in case if more than one object reside on the same server and need to be presented simultaneously. The synchronization between objects is provided by message passing techniques between threads. For each PoA, you can notice that the plan is generated as a combination of mainly four methods: *WaitMessage*, *SendMessage*, *ApplyQoSRules* and *SendObject*. These methods work as follows.

1) *WaitMessage (srvAddr , msgType)*: This is a blocking method that blocks its caller until it receives a message whose type is *msgType* from the server whose address is *srvAddr*.
2) *ApplyQoSRules (mediaObj)*: This method probes the network between the caller and the client and make a decision of applying the QoS rules, based on the result of network probing and the type of the media object *mediaObj* itself. The application of the QoS rules is performed by the server on which the media object is residing.
3) *SendObject (cltAddr, mediaObj)*: This method sends the *mediaObj* to the client whose address is *cltAddr*, while applying the QoS rules decided by the *ApplyQoSRules* method.
4) *DoAction (cond, srvAddr, actionType)*: If condition *cond* is *true*, the method asks the server whose address is *srvAddr* , to do an action of type *actionType*. As a result of calling this method, a thread is created that receives the data sent from the server as a result of the requested action and processes this data, by calling the appropriate media player. Once the action is terminated, a notification will be sent back to the client's main thread.

As can be noticed from the PoAs for SERVER_1 and SERVER_2, holding the initial objects to be played, they start by waiting for a message from the client, holding the media players, to start playing objects A and B.

Notifications of the "end of presentation" of the media objects are handled by the client's scheduler. The scheduler can be implemented as a combination of *singleton* and *observer* design patterns. The singleton pattern is used to define the scheduler object, since we need one instance of scheduler per multimedia presentation at the client side. The observer pattern can be used in implementing the notification mechanism, where the media object presentation end instants are defined as events, and the client can subscribe as an observer of these events.

Communication among threads on servers and the client can be achieved by socket-based communication, which enables each thread as well as the client to deal with networking as if it is a file input/output operation. So, any thread or client can read/write from/to a socket. These sockets can be used to: (1) carry synchronization messages between the servers and the client, (2) transfer adapted multimedia objects from servers to the client, and (3) transfer access control commands, such as start, stop, pause and resume, from client to servers.

The whole system of distributed QoS handler can be implemented using the Service Oriented Architecture (SOA) [11] where the QoS adaptation tasks will be considered as the services that are provided and executed on the servers, and, consequently, the client's task will be limited to schedule when the media objects need to be adapted, and then play the received, adapted media object(s).

```
// Start of PoA for CLIENT
MainThread {
end_A = DoAction (true, SERVER_1, start_A);
end_B = DoAction (true, SERVER_2, start_B);
end_C = DoAction (end_A∧end_B, SERVER_3,
start_C);
end_D = DoAction (end_C, SERVER_1, start_D);
}
// End of PoA for CLIENT
```

Fig. 4 PoA for CLIENT

```
// Start of PoA for SERVER_1
Thread_1_1 {
WaitMessage (CLIENT, start_A);
ApplyQoSRules (A);
SendObject(CLIENT, A);
}

Thread_1_2 {
WaitMessage (SERVER_3, start_D);
ApplyQoSRules (D);
SendObject (CLIENT, D);
}
// End of PoA for SERVER_1
```

Fig. 5PoA for SERVER_1

```
// Start of PoA for SERVER_2
Thread_2_1 {
WaitMessage (CLIENT, start_B);
ApplyQoSRules (B);
SendObject(CLIENT, B);
}
// End of PoA for SERVER_2
```

Fig. 6PoA for SERVER_2

```
// Start of PoA for SERVER_3
Thread_3_1 {
WaitMessage (CLIENT, start_C);
WaitMessage (CLIENT, start_C);
ApplyQoSRules (C);
SendObject(CLIENT, C);
}
// End of PoA for SERVER_3
```

Fig. 7PoA for SERVER_3

V. CONCLUSION

In this paper, we presented a mechanism for an application level distributed QoS handler that allows servers to act in an autonomous way in taking QoS decisions and performing QoS actions, while sending the media objects data to the client. So, the application of the QoS rules for adapting the media objects are distributed among the servers on which the media objects are residing. In this case, the client is completely freed from the QoS-related tasks and it focuses only on playing the received media objects and notifying the end of their presentation to the concerned servers. We claim that this solution results provides a better load balance for playing a distributed multimedia application, where the tasks of applying QoS rules and playing the objects are better distributed among the playing client and the media object servers.

ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 4, June 2013

REFERENCES

[1] A.R. Abo Elenien, **L. S. Ismail**, and H. S. Bedor (2004), 'Quality of service handler for MPEG video in best effort environment', Proceedings of the International Conference on Electrical, Electronics and Computer Engineering (ICEEC'04), IEEE, Egypt, p.p. 393-398, Egypt, September 2004.

[2] D.Bulterman (2009), 'Basic SMIL Timing', SMIL 3.0: Flexible Multimedia for Web, Mobile Devices and Daisy Talking Books, 2nd Edition, Springer, pp. 125-156.

[3] N. Layaida, C. Roisin and **L. Sabry Ismail** (2011), 'Chapitre 6: Support d'exécution de documents multimédia', Systemesmultimédia communicants, Hermès Science Publications, Paris, .

[4] M. Sadallah, O. Aubert, and Y. Prie (2011), 'Hypervideo and Annotations on the Web', Proceedings of the 2011 Workshop on Multimedia on the Web (MMWeb 2011), IEEE, p.p. 10-15, Austria, September 2011.

[5] L. Yulong and L. Naiwen (2012), "A multimedia synchronization MDTSPN model based on Petri nets", Proceedings of the 2012 Eighth International Conference on Natural Computation (ICNC 2012), IEEE, p.p. 497-501, May 2012.

[6] W3C (2008), "Extensible Markup Language (XML) 1.0 (Fifth Edition)", Internet: http://www.w3.org/TR/REC-xml/ , November 2008, [accessed on June 10, 2013].

[7] W3C (2004), "Extensible Markup Language (XML) 1.1", Internet: http://www.w3.org/TR/2004/REC-xml11-20040204/#NT-doctypedecl , February 2004, [accessed on June 10, 2013].

[8] W3C (2012), "W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures", Internet: http://www.w3.org/TR/xmlschema11-1/ , April 2012, [accessed on June 10, 2013].

[9] E. Van der Vlist (2011), 'What RELAX NG Offers?',RELAX NG, Kindle Edition, O'Reilly, pp. 3-7.

[10] W3C (2007), "Extensible Stylesheet Language (XSL) Version 1.1", Internet: http://www.w3.org/TR/2007/REC-xslt20-20070123/ , January 2007, [accessed on June 10,2013].

[11] M. Bell (2010), 'Service-Oriented Discovery and Analysis Road Map Patterns',SOA Modeling Patterns for Service Oriented Discovery and Analysis, 1st Edition, Wiley, pp. 21-68.