HAL
open science

Semantic multimedia remote viewer for collaborative mobile thin clients

Bojan Joveski

To cite this version:

Bojan Joveski. Semantic multimedia remote viewer for collaborative mobile thin clients. Other [cs.OH]. Ecole Nationale Supérieure des Mines de Paris, 2012. English. NNT : 2012ENMP0100 . pastel-00873162

HAL Id: pastel-00873162

https://pastel.hal.science/pastel-00873162

Submitted on 15 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

MINES ParisTech

École doctorale n°432: Sciences des Métiers de l'Ingénieur

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

l'École nationale supérieure des mines de Paris

Spécialité "Informatique temps réel, robotique et automatique"

présentée et soutenue publiquement par

Bojan JOVESKI

le 18 Décembre 2012

Dispositif de rendu distant multimédia et sémantique pour terminaux légers collaboratifs

Directeur de thèse : **Françoise PRETEUX**
Co-directeur de thèse : **Mihai MITREA**

Jury

M. Michel JOURLIN, Professeur, LHC UMR CNRS, Université de Saint-Etienne	Rapporteur
M. Touradj EBRAHIMI, Professeur, MSP , Swiss Federal Institute of Technology	Rapporteur
M. Jean-Noël PATILLON, PhD, DTSI, CEA Saclay	Examinateur
M. Bart DHOEDT, Professeur, IBCN, Ghent University - iMinds	Examinateur
M. Najah NAFFAH, PhD, Executive director, Prologue	Examinateur
Mme. Françoise PRETEUX,Professeur, Direction des Recherches, Mines ParisTech	Examinateur
M. Mihai MITREA, HDR, ARTEMIS, Mines-Telecom, Telecom SudParis	Examinateur
M. Ian James MARSHALL, Chef cellule d'innovation, Prologue	Invité

T H È S E

MINES ParisTech
Mathématiques et Systèmes, CAOR
60, Boulevard Saint-Michel 75272 PARIS cedex 06, France

I dedicate this thesis to my wife Mladena,

for being my pillar, my joy and my guiding light.

Acknowledgments

I am deeply indebted to my thesis director Professor Françoise Prêteux, for giving me an opportunity to work in this challenging research topic as part of the ARTEMIS department at Institut Mines Telecom, Telecom SudParis, and as PhD student at the CAOR department at MINES ParisTech. I thank her not only for the academic support but also for promoting and supporting me in the world of MPEG standardization.

I would like to express my sincere gratitude to my thesis co-director HDR Mihai Mitrea, who offered his continuous advice and encouragement throughout the course of this thesis. I thank him for the systematic guidance, great effort he put into training me in the scientific field, good teaching, good company and lots of good ideas. Without him this thesis would not have come to a successful completion with his continuous support and help to remain focused for achieving my goal.

I am grateful to Professor Michel Jourlin at Laboratoire Hubert Curien, Université Jean Monnet Saint-Etienne and Professor Touradj Ebrahimi at MSP Group, Swiss Federal Institute of Technology (EPFL), for granting me the honor and accept the task of reviewing this thesis. I thank them for their precious comments and suggestions for its amelioration and perspectives of this work. I would also like to thank Professor Bart Dhoedt at INTEC-IBCN, Ghent University for the collaborative work, evaluation of the thesis and the honor of presiding the jury. I would also like to thank PhD Jean-Noël Patillon at CEA LIST and PhD Najah Naffah, Executive director at Prologue SA, for the time they spent in the evaluation of the thesis with thoughtful comments and advices for the future of this work in the industry field.

I would like to express my sincere thanks to Iain-James Marshall, Responsible for the innovation unit at Prologue SA, for generously sharing his time in our cooperative MPEG work and for accepting this thesis as part of the jury. Many thanks for his native English language support that helped me improve my language skills.

I thank Professor Arnaud De La Fortelle, director of the CAOR department and Professor François Goulette, coordinator of the thesis program at CAOR, for welcoming me at MINES ParisTech and for their valuable feedback provided during my mid-term presentation.

I thank Mrs. Evelyne Taroni at ARTEMIS Department, Institut Mines Telecom, Telecom SudParis, Mrs. Christine Vignaux at CAOR, MINES ParisTech and Mrs. Sylvie Barizzi-Loisel at MINES ParisTech for their patience and valuable help in the administration matters.

My colleagues Ludovico Gardenghi and Rama Rao Ganji, deserve a special mention. I thank them for helping me with their software development skills and their availability during this thesis.

I thank PhD Pieter Simoens at Ghent University and Abdeslam Taguengayte, Research and Development Team Manager at Prologue SA, for our cooperative research work, reported in several publications.

Special thanks to my colleagues: Margarete Ortner, Afef Chammem and Adriana Garboan, for their understanding and generous help not only as colleagues but also as friends. I thank them for the positive spirit they brought in all these years.

Last but not least, I wish to take this opportunity to thank my parents, who raised me, taught me, and loved me and to my brother Filip for his intellectual and motivational discussions and emotional support throughout my academic career.

Abstract

In accordance with current day user expectancies, no functional discrepancies should be noticeable between mobile thin client and fixed desktop applications.

The *thin client paradigm* covers all the scientific, technical and applicative issues related to a terminal (desktop, PDA, smartphone, tablet) essentially limited to I/O devices (display, user pointer, keyboard) and with its all computing and storage resources located on a remote server farm. This model implicitly assumes the availability of a network connection (be it wired or wireless) between the terminal and the computing resources.

From the architectural point of view, the thin client paradigm can be accommodated by a client-server model, where the client is connected to the server through a connection managed by a given protocol. From the functional point of view, the software application (text editing, www browsing, multimedia entertainment, …) runs on the server and generates some semantically structured graphical output (*i.e.* a collection of structured text, image/video, 2D/3D graphics, …). This graphical content should be transmitted and visualized by the client, where the user interactivity is captured and sent back to the server for processing.

The term *remote viewer* refers to all software modules, located at both end points (server and client), making possible the graphical content generated by the server to be displayed at the client side and the subsequent user events to be sent back to the server, in real time.

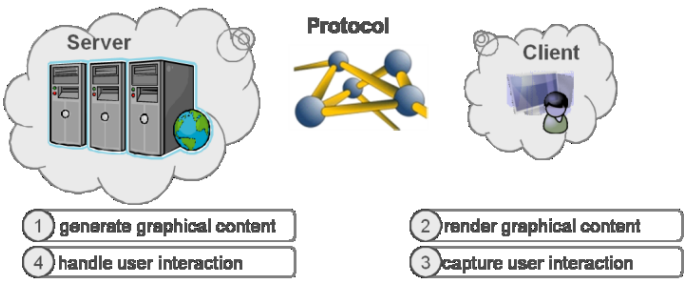


Figure 1. Multimedia remote viewer

Defining of a multimedia remote viewer for mobile thin clients remains a challenging research topic, coming across with threefold scientific/technical constraint relating to the user expectancies, the underlying mobile environment issues and the market acceptance. First, on the user expectancies side, the remote viewer should provide at the client side heterogeneous multimedia content and the support for ultimate collaboration functionalities. Second, from the mobility point of view, issues related to the network (arbitrarily changing bandwidth conditions, transport errors, and latency) and to the terminal (limitations in CPU, storage, and I/O resources)

should be addressed. Finally, the market acceptance of such a solution depends on its ability of featuring terminal independency and of benefiting from community support.

Current day remote viewer solutions for mobile thin clients are inherited from wired environments, where several reference technologies are available for decades: X, VNC, NX, RDP, to mention but a few. Regardless of its original type, the heterogeneous graphical content (text, image, graphics, video, 3D, …) generated by the server is converted into sequences of images (eventually a mixture of images and graphics), which are then interactively displayed by the client. Such an approach would appear to be inappropriate when addressing the above-mentioned mobile thin client constraints. First, it prevents the client from having a true multimedia experience and offers no support for collaboration (which is supposed to be solved by additional devoted mechanisms). Second, it considers the multimedia content adaptive compression solely from the particular point of view of image compression, thus resulting in sub-optimal network resource consumption. Finally, these solutions depend on the terminal hardware/software peculiarities, thus representing a pitfall for a standard deployment on the market.

The present thesis follows a different approach and introduces a semantic multimedia remote viewer for collaborative mobile thin clients, see Table 1. The principle is based on representing the graphical content generated by the server as an interactive multimedia scene-graph, enriched with novel components for direct handling (at the content level) of the user collaboration. In order to cope with the mobility constraints, a semantic scene-graph management framework was design (patent pending) so as to optimize the multimedia content delivery from the server to the client, under joint bandwidth-latency constraints in time-variant networks. The compression of the collaborative messages generated by the users is done by advancing a devoted dynamic lossless compression algorithm (patented solution). This new remote viewer was evaluated incrementally by the ISO community and its novel collaborative elements are currently accepted as extensions for the ISO IEC JTC1 SC 29 WG 11 (MPEG-4 BiFS) standard.

The underlying software demonstrator, referred to as MASC (Multimedia Adaptive Semantic Collaboration), is implemented as open-source. The solution was benchmarked against its state-of-the-art competitors provided by VNC (RFB) and Microsoft (RDP).

It was demonstrated that: (1) it features high level visual quality, *e.g.* PSNR ranges between 30 and 42dB or SSIM has values larger than 0.9999; (2) the downlink band-width gain factors range from 2 to 60 while the up-link bandwidth gain factors range from 3 to 10; (3) the network round-trip time is reduced by factors of 4 to 6; (4) the CPU activity is larger than in the Microsoft RDP case but is reduced by a factor of 1.5 with respect to the VNC RFB.

The MASC is evaluated for its potential industrialization in various applicative fields, such as application virtualization in the cloud (in partnership with Prologue), promoting collaborative decision making system for video surveillance applications (in partnership with CASSIDIAN) as well as virtual collaborative environment for medical assistance (in partnership with Philips HealthCare and Bull).

Table 1. Collaborative mobile thin clients: constraints, challenges, state of the art limitations and thesis contributions

Constraints	Challenge	Current limitation	Thesis contributions
User Expectancies			
Multimedia	True multimedia content on the client side	Image (sometimes image&graphics)	Multimedia scene-graph architecture, ensuring a hierarchical composition of the content (text, graphics, image, video, 3D, …) *(journal publication)*
Collaborative experience	Full collaboration support	No support at the content level, dedicated mechanism	Specification and standardization of the collaboration elements for the multimedia scene-graph *(ISO IEC JTC1 SC 29 WG 11 - MPEG-4 BiFS version 2)*
Mobility			
Time-variant network bandwidth & latency	Real-time compression algorithm for multimedia content	*Downlink:* • compression algorithm based on regions of interest in images	*Downlink:* • real-time scene-graph adaptation mechanism *(patent pending)* • semantic management of the multimedia compression algorithms
		Uplink: •static compression for user messages	*Uplink.* • dynamic encoding method for collaboration messages *(patent pending)*
Market acceptance			
Terminal/OS proliferation	Terminal independency	Terminal/OS dependent solutions	ISO standard compatibility
Community support	Open source	Proprietary vs. open source	Open source software architecture

Table of contents

Acknowledgments iii
Abstract vii
Table of contents xi
List of figures xiii
List of tables xv
List of code xvii
Chapter 1. Introduction 1
1.1 Context 2
1.1.1. Online social networking 2
1.1.2. Cloud computing 5
1.1.3. Bridging social networking and cloud computing 8
1.2 Objectives 8
1.3 Thesis structure 11
Chapter 2. State of the art 13
2.1 Content representation technologies 14
2.1.1. Introduction 14
2.1.2. Comparison of content representation technologies 15
2.1.3. BiFS and LASeR principles 17
2.1.4. Conclusion 24
2.2 Mobile Thin Clients technologies 25
2.2.1. Overview 25
2.2.2. X window system 25
2.2.3. NoMachine NX technology 28
2.2.4. Virtual Network Computing 30
2.2.5. Microsoft RDP 32
2.2.6. Conclusion 34
2.3 Collaboration technologies 36
2.4 Conclusion 41

Chapter 3. Advanced architecture 43
3.1 Functional description 44
3.2 Architectural design 46
3.2.1. Server-side components 47
3.2.2. Client-side components 60
3.2.3. Network components 62
3.3 Conclusion 64
Chapter 4. Architectural benchmark 65
4.1 Overview 66
4.2 Experimental setup and results 67
4.2.1. Visual quality 67
4.2.2. Down-link bandwidth consumption 70
4.2.3. User interaction efficiency 73
4.2.4. CPU activity 74
4.3 Discussions 76
4.3.1. Semantic Controller performance 76
4.3.2. Pruner performance 77
Chapter 5. Conclusion and Perspectives 79
5.1 Conclusion 80
5.2 Perspectives 81
List of abbreviations 85
References 87
Appendix I 91
Appendix II 93
Appendix III 95
List of publications 117

List of figures

Figure 1. Multimedia remote viewer vii
Figure 1.1. Facebook popularity, registered active users during the period from 2004 till 2011 3
Figure 1.2. YouTube popularity, videos viewed daily expressed in billions 3
Figure 1.3. Wikipedia articles submitted yearly from 2001 till 2012 4
Figure 1.4. Total internet exchange in only 1 minute 4
Figure 1.5. Cloud computing at a glance 5
Figure 1.6. Active EC2 virtual machines grouped by operating system 6
Figure 1.7. Total market share by operating systems in 2012 6
Figure 1.8. Total market share by cloud computing solutions in 2011 7
Figure 1.9. The workloads processed in the cloud will reach more than 50% by 2015 7
Figure 1.10. The traffic exchanged in the cloud per year by 2015 (1 ZB = 270 Bytes) 7
Figure 1.11. Collaborative mobile thin clients: a new generation of mobile thin clients bridging the gap between the cloud computing and communitarian users 9
Figure 2.1. Scene technology support for mobile thin clients 14
Figure 2.2. Concurrent solutions for heterogeneous content encoding, updating and streaming 17
Figure 2.3. BiFS scene-graph description example 19
Figure 2.4. LASeR architecture 23
Figure 2.5. X windows system server-client architecture 25
Figure 2.6. X window system server and client side content rendering 26
Figure 2.7. NoMachine architecture 28
Figure 2.8. NX client-server rendering 29
Figure 2.9. The VNC server-client architecture 30
Figure 2.10. VNC server-client content rendering 31
Figure 2.11. Windows RDP server-client architecture 32
Figure 2.12. Microsoft RDP server-client content rendering 33
Figure 2.13. Comparison of the current remote display solutions 35
Figure 2.14. The concept of real-time user collaboration 36
Figure 2.15. Creating a text document using the Google Docs 37
Figure 2.16. World of Warcraft (WoW) screenshot 38
Figure 2.17. Video streaming on YouTube using www browser 39
Figure 2.18. Video conferencing using Skype 39
Figure 3.1. State-of-the-art architectural framework for mobile thin client remote display 44
Figure 3.2. Advanced architectural framework for mobile thin client remote display 44

Figure 3.3. Detailed architectural framework 46

Figure 3.4. Flowchart for image management 50

Figure 3.5. The Pruning mechanism, exemplified for image content 52

Figure 3.6. Traditional scene-graph creation 52

Figure 3.7. Advanced scene-graph adaptively created 53

Figure 3.8. General architecture for collaborative scenes 57

Figure 3.9. Direct client to client collaboration by using the collaboration node 62

Figure 4.1. Illustration of the text editing application run on the server (a) and displayed on the mobile thin client, after its conversion into BiFS / MASC-BiFS (b) and MASC-LASeR (c) 68

Figure 4.2. Illustration of the www browsing application run on the server (a) and displayed on the mobile thin client, after its conversion into BiFS / MASC-BiFS (b) and MASC-LASeR (c) 68

Figure 4.3. Average bandwidth consumption (in KBytes) for text editing (gEdit), as a function of time 71

Figure 4.4. Average bandwidth consumption (in KBytes) for www browsing (Epiphany), as a function of the browsing step 71

Figure 4.5. The average maximum CPU consumption (in %) while browsing, as a function of the browsing step ... 75

Figure 4.6. Performance of the Semantic Controller block: total bandwidth consumption in the case of text editing over image type (encoding used for the images in the scene-graph) 76

Figure 4.7. Performance of the Semantic Controller block: total bandwidth consumption in the case of www browsing (executing the 9 steps) over image type (encoding used for the images in the scene-graph) 77

Figure 4.8. Average maximum CPU activity as a function of time expressed in seconds, in the text editing experiment 78

Figure 4.9. Average maximum CPU activity as a function of time expressed in seconds, in the www browsing experiment 78

Figure 5.1. Enablers for the advanced collaborative mobile thin client framework 80

Figure 5.2. Extension from Linux to Windows applications 83

Figure 5.3. Beyond MPEG collaboration – cross standard collaboration 83

List of tables

Table 1. Collaborative mobile thin clients: constraints, challenges, state of the art limitations and thesis contributions ix

Table 1.1. Thesis objectives 10

Table 2.1. Current collaboration status 40

Table 2.2. Illustrations of the current limitations of the existing technologies 41

Table 3.1. Traditional MPEG-4 scene-graph processing 54

Table 3.2. Advanced scene-graph adaptation 55

Table 4.1. Visual quality evaluation for X to MPEG (BiFS, MASC-BiFS and MASC-LASeR) conversion 70

Table 4.2. Average overcharge traffic (in KB) induced by network disconnection, for text editing 73

Table 4.3. Average overcharge traffic (in KB) induced by network disconnection, for www browsing. 73

Table 4.4. The size of the traffic generated through the back channel by elementary user events 74

List of code

Code 2.1. Example of BiFS scene-graph, represented using VRML 22
Code 2.2. LASeR scene example of the Figure 4.3, including SAF aggregation 24
Code 2.3. X Protocol request description, polySegment 27
Code 2.4. X Protocol request description, putImage 27
Code 2.5. X Protocol request description, polyText16 27
Code 2.6. Code sample written in C languge, for compressing the polySegment X request 29
Code 2.7. Graphical primitive, used by VNC server 31
Code 2.8. VNC HEXTILE encoding function expressed in C language 31
Code 2.9. VNC PIXEL FORMAT encoding function expressed in C language 32
Code 2.10. Binary description of an RDP rectangle message 33
Code 2.11. Binary description of an RDP image pixel message 34
Code 3.1. X Protocol description of polyRectangle syntax 48
Code 3.2. Syntax of parsing polyRectangle by the XParser 48
Code 3.3. XML description of BiFS conversion of a rectangle 49
Code 3.4. SVG description of LASeR conversion of a rectangle 49
Code 3.5. Part of the C language code enriching the scene-graph with JavaScript functionality for mouse click 57
Code 3.6. Technical description of the CollaborationNode 58
Code 3.7. C language code for posting the mouse left button click on the application 59

Chapter 1. Introduction

Tout d'abord, ce chapitre introduit les définitions liées au paradigme du terminal léger et identifie le rôle que les systèmes de rendu distants jouent dans un tel cadre, ainsi que leur limitations en enjeux. Ensuite, les principales contributions de la thèse sont succinctement présentées. Finalement, la structure de la thèse est précisée.

1.1 Context

In August 2008, the number of users accessing various online social networks stayed quite modest (Facebook – 100 million, MySpace – 300 million, Tweeter – 5 million), the number of mobile connected Internet users did not reached yet the limit of 150 million and cloud computing was rather a concept then a business *per-se* (Google Apps and Web 2.0 just emerged on the market).

Nowadays, Facebook approaches the 1 billion users threshold, 6 billion of mobile devices are Internet connected and cloud computing generates 109 billion of dollars in revenue a year, [Gartner, 2012].

By 2020, the social networking will cover 70% of the Earth population, number of mobile connected Internet devices will be multiplied by 2 and the cloud computing revenue by 3, [GSMA, 2012].

However, despite the synchronicity in their explosive development, the on-line networking and cloud computing revolutions followed different ways.

1.1.1. Online social networking

Social networking, also referred to as social Internet media, encompasses the Internet-based tools that make easier for connected users to share (watch, listen and interact with) any type of multimedia content.

Internet leverages the social networking to the level of becoming the most intensive business and marketing platform. Individual users, powered by heterogeneous mobile/fixed terminals, benefiting from the open standards open source software, aggregate themselves into online social networking in order to collaboratively enjoy a new type of multimedia experience.

Nowadays, 9 billion of devices are connected to Internet and their number is forecasted to reach 24 billion by 2020 [GSMA, 2011].

In order to answer to the interests of the all online communities, plenty of online tools are currently supporting social networking: Facebook, Youtube, mySpace, Flickr, Google+, Hi5, Tweeter.

Facebook

Launched in February 2004, with the initial idea of exchanging information between the connected users, in just a couple of years it served several million of users. By offering to the users the possibility to exchange images, audio, video, and text, Facebook represents today the world most exploited social network, registering 950 million of online users. In only one minute, 135 000 photos are uploaded, 75 000 events are created and 100 000 demands for user

interconnections are sent [Facebook, 2012], see Figure 1.1. This huge amount of information brings to light the need for intelligent cloud computing platforms.

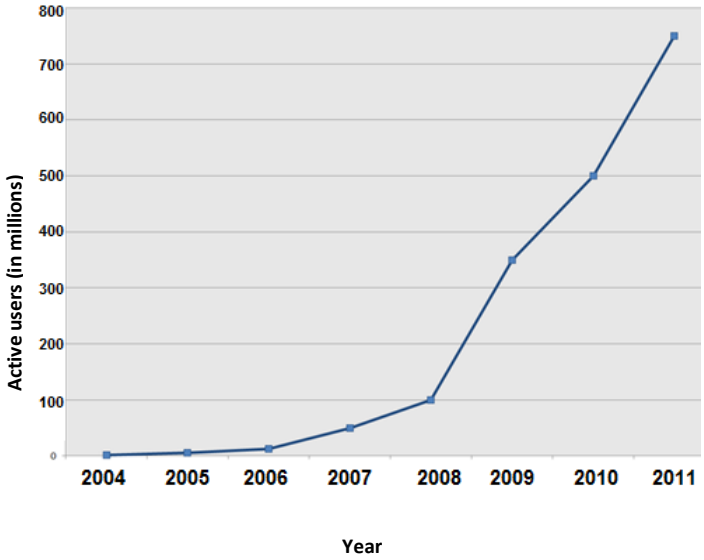


Figure 1.1. Facebook popularity, registered active users during the period from 2004 till 2011

YouTube

The challenge to enable the online users to share their videos trough Internet was achieved by YouTube in 2005: online users can upload, share and comment their videos. This simple, still novel at that time online tool, has been very easy exploited by millions of users since then. Today, 72 hours of video are upload each minute, more than 4 billion hours of videos are watched per month by 800 million visits (per month) [Youtube, 2012], see Figure 1.2.

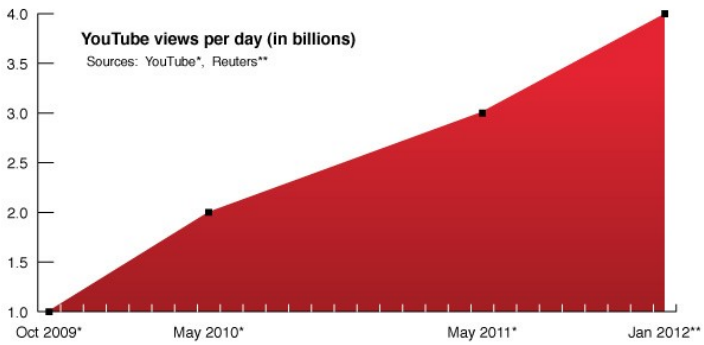


Figure 1.2. YouTube popularity, videos viewed daily expressed in billions

Wikipedia

Founded in 2001, Wikipedia becomes the nonprofit multilingual internet encyclopedia, translated in 285 languages. With the help of 100 000 active contributors, it accommodates 23 million articles, and serves the online users with more than 2.7 billion monthly page views, [Wikipedia, 2012], see Figure 1.3.

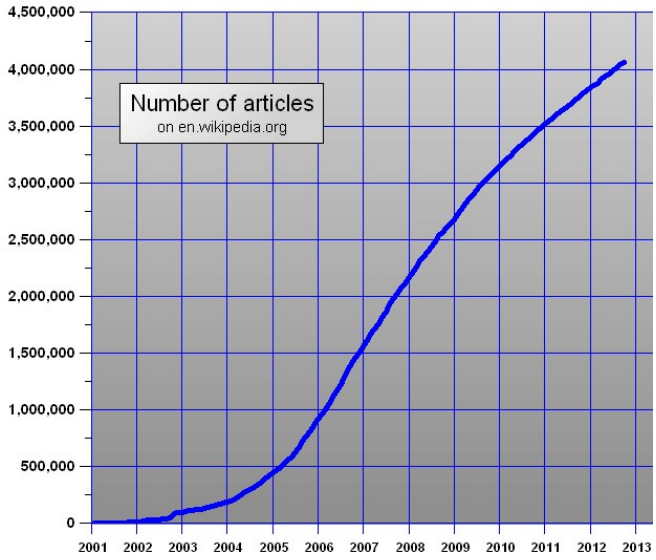


Figure 1.3. Wikipedia articles submitted yearly from 2001 till 2012

Conclusion

These basic three examples, although different by their finality, exemplify the social networking main characteristics:

- a continuous grow, by registering each day new users and by involving existing users in new experiences;
- although the social networking tools are continuously updated/replaced/changed, the user social interest remains;
- the exchanged data reaches more than 650TB a minute (out of the total of 2025 TB exchanged in Internet each minute), see Figure 1.4, [Intel, 2012].

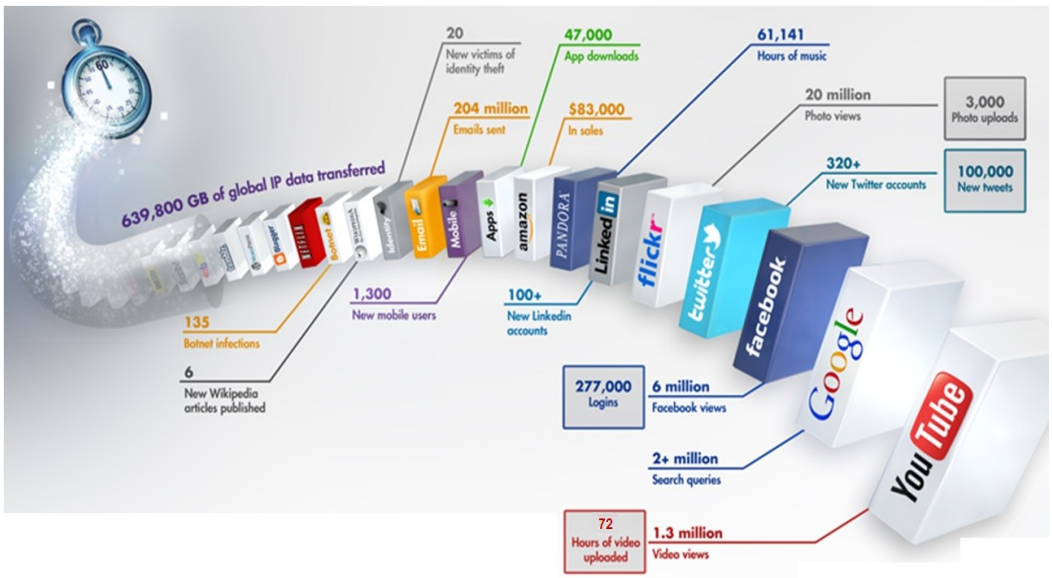


Figure 1.4. Total internet exchange in only 1 minute

Under this framework, cloud computing positioned itself as very appealing solution for an intelligent and powerful data management system for social networking data in particular and for any type of data in Internet, in general.

1.1.2. Cloud computing

With the release of the "Elastic Compute Cloud" by Amazon (2006), "Microsoft Azure" by Microsoft (2010), and "Open Cloud Computing Interface" by Open Source community (2010) the new era of cloud computing platforms was started. These huge industrial supports raised the cloud computing to the ultimate working environment, where proprietary software applications (Office, Photoshop, 3D Max, …), running under their traditional OS (Microsoft Windows, Apple Lion, Linux Ubuntu, …) would give the user access to any kind of Big data (documents, photography/video archives, medical records, …), see Figure 1.5.

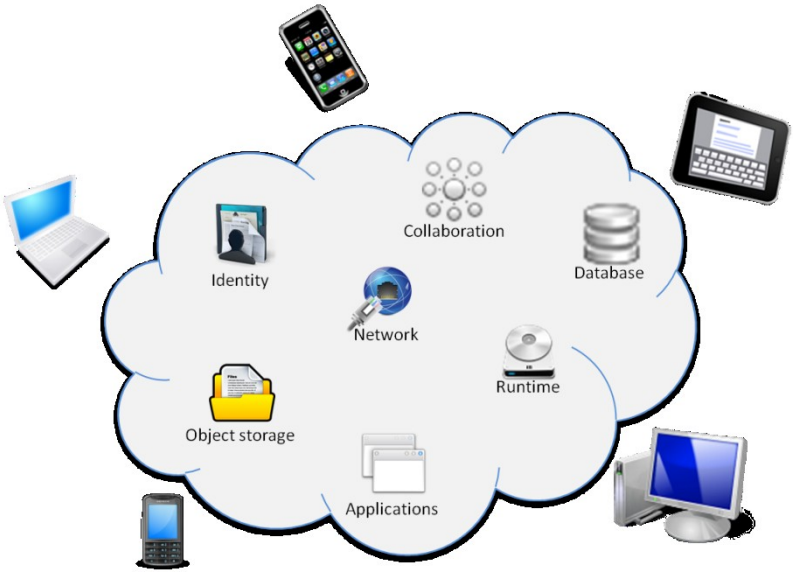


Figure 1.5. Cloud computing at a glance

Amazon cloud

Elastic Compute Cloud (EC2) released by Amazon for a limited public in 2006, is one of the first virtual computing environments allocating the hardware resources dynamically, allowing the online users to rent virtual computers by using Internet.

Today EC2 is capable to manage a large variety of operating systems, load custom applications, manage network access permissions and run software in real-time, according to the users requirements. It accommodates more then 46 000 active virtual machines, running multiple OSs, Figure 1.6. It can be seen that the various Linux distributions (Ubuntu, Linux, RedHat, Debian, Fedora,…) cover more than 80% of them.

Figure 1.6. Active EC2 virtual machines grouped by operating system

Microsoft cloud

With the release of Windows Azure in 2010, Microsoft offered to the users a cloud computing platform to build, deploy and manage applications. As nowadays the Microsoft Windows OS dominates the local setups application environments (more than 84% from the total, [Market, 2012]), Windows Azure has a huge potential in future exploitations, see Figure 1.7.

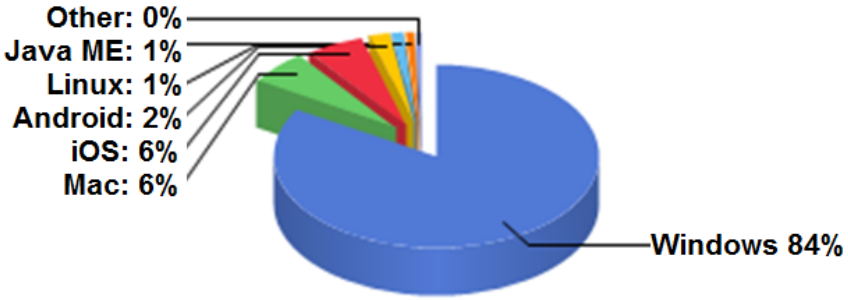


Figure 1.7. Total market share by operating systems in 2012

VMware cloud

In 2008 VMware announced an integrated solution for building and managing a complete cloud infrastructure. By the middle of 2009 VMware releases the vSphare 4, leveraging addition of virtual disks and on the fly. This solution provides all infrastructure services necessary to make workloads operational in minutes. By offering to these services, vSphere of VMware becomes the most distributed tool by 2011 having 89% of the market share, see Figure 1.8.

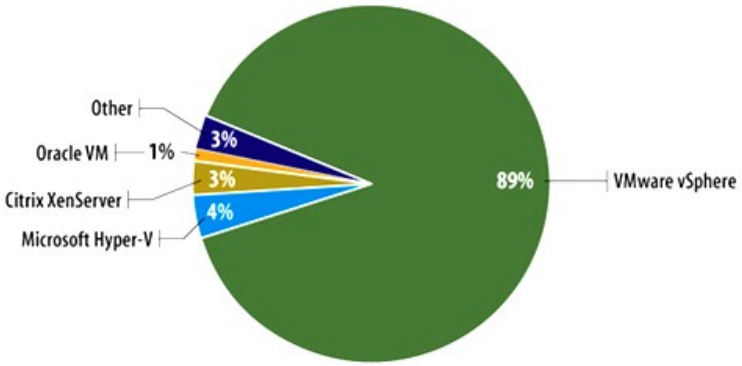


Figure 1.8. Total market share by cloud computing solutions in 2011

Conclusion

This migration from fixed setups to cloud is expected to grow in the very next years, see Figure 1.9 and Figure 1.10: by 2015, more than 50 percent of all processing workloads will take place in the cloud, and the data exchange will reach 4.8 Zettabytes, [Cisco, 2012].

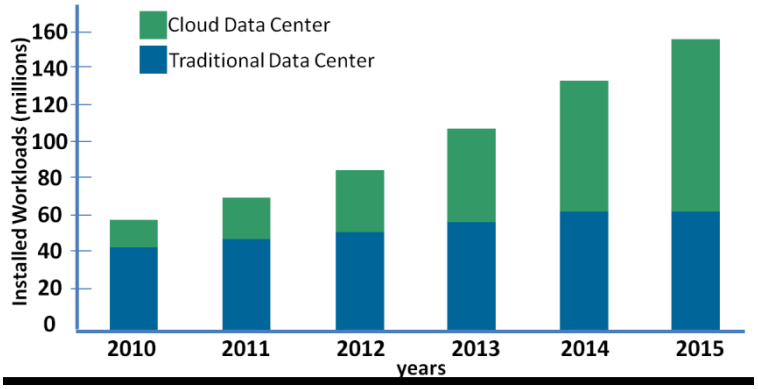


Figure 1.9. The workloads processed in the cloud will reach more than 50% by 2015

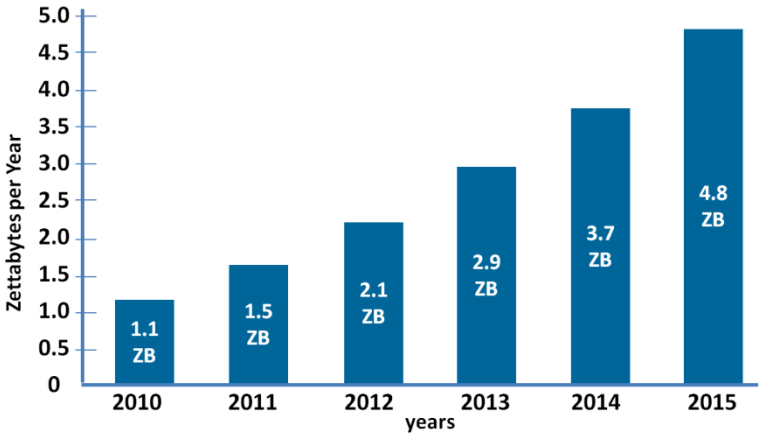


Figure 1.10. The traffic exchanged in the cloud per year by 2015 (1 ZB = 2^{70} Bytes)

1.1.3. Bridging social networking and cloud computing

The gap between social networking and cloud computing can be bridged by developing remote viewer solutions.

In the widest sense, the *thin client* paradigm refers to a terminal (desktop, PDA, smartphone, tablet) essentially limited to I/O devices (display, user pointer, keyboard), with all related computing and storage resources located on a remote server farm. This model implicitly assumes the availability of a network connection (be it wired or wireless) between the terminal and the computing resources.

Within the scope of this thesis, the term *remote display* refers to all the software modules, located at both end points (server and client), making possible, in real time, for the graphical content generated by server to be displayed on the client end point and for subsequent user events to be sent back to the server. When these transmission and display processes consider, for the graphical content, some complementary semantic information (such as its type, format, spatio-temporal relations or usage conditions to mention but a few) the remote display then becomes a *semantic remote display*[1].

Our study brings to light the potential of *multimedia scene-graphs* for supporting semantic remote displays. The concept of the scene-graph emerged with the advent of the modern multimedia industry, as an attempt to bridge the realms of structural data representation and multimedia objects. While its definition remains quite fuzzy and application dependent, in the sequel we shall consider that a scene-graph is [BiFS, 2006]: "*a hierarchical representation of audio, video and graphical objects, each represented by a […] node abstracting the interface to those objects. This allows manipulation of an object's properties, independent of the object media.*" Current day multimedia technologies also provide the possibility of direct interaction with individual nodes according to user actions; such a *scene-graph* will be further referred to as an *interactive multimedia scene-graph*.

In order to allow communitarian users to benefit from the cloud computing functionalities, a new generation of thin clients should be designed. They should provide universal access (any device, any network, any user …) to virtual collaborative multimedia environments, through versatile, user-friendly, real time interfaces based on open standard and open source software tools.

1.2 Objectives

The main objectives when developing a mobile thin client framework is to have the same user experience as when using a fixed desktop applications, see Figure 1.11.

[1] The usage of the word *semantic* in this definition follows the MPEG-4 standard specification [BiFS, 2006] and the principles in some related studies [Asadi, 2005], [Izquierdo, 2003].
[2] Although the scene elements are structured in a tree, the standard name is the *scene-graph*.
[3] When computing the confidence intervals, the correlation between the images corresponding to successive scene updates was

Figure 1.11. Collaborative mobile thin clients: a new generation of mobile thin clients bridging the gap between the cloud computing and communitarian users

Under this framework, the definition of a multimedia remote display for mobile thin clients remains a challenging research topic, requiring at the same time a high performance algorithm for the compression of heterogeneous content (text, graphics, image, video, 3D, …) and versatile, user-friendly real time interaction support [Schlosser,2007], [Simoens, 2008]. The underlying technical constraints are connected to the network (arbitrarily changing bandwidth conditions, transport errors, and latency), to the terminal (limitations in CPU, storage, and I/O resources), and to market acceptance (backward compatibility with legacy applications, ISO compliance, terminal independence, and open source support).

The present thesis introduces a semantic multimedia remote viewer for collaborative mobile thin clients, see Table 1.1. The principle consists of representing the graphical content generated by the server as an interactive multimedia scene-graph enriched with novel components for directly handling (at the content level) the user collaboration. In order to cope with the mobility constraints, a semantic *scene-graph* management framework was design (patent pending) so as to optimize the multimedia content delivery from the server to the client, under joint bandwidth-latency constraints in time-variant networks. The compression of the collaborative messages generated by the users is done by advancing a devoted dynamic lossless compression algorithm (patented solution). This new remote viewer was incrementally evaluated by the ISO community and its novel collaborative elements are currently accepted as extensions for ISO IEC JTC1 SC 29 WG 11 (MPEG-4 BiFS) standard [BiFS, 2012].

Table 1.1. Thesis objectives

Constraints	Challenge
User Expectancies	
Multimedia	True multimedia content on the client side
Collaborative experience	Full collaboration support
Mobility	
Time-variant network bandwidth & latency	Real-time compression algorithm for multimedia content
Market acceptance	
Terminal/OS proliferation	Terminal independency
Community support	Open source

1.3 Thesis structure

The thesis structure is divided in three main chapters.

Chapter 2 represents threefolded analysis of the state-of-the-art technologies encompassed by the remote viewers. In this respect, Section 2.1 investigates the use of multimedia content when designing a remote viewer. The existing technologies are studied in terms of binary compression, dynamic updating and content streaming. Section 2.2 considers the existing wired or wireless remote viewing solutions and assess their compatibility with the challenges listed in Table 1.1. Section 2.3 brings to light the way in which the collaborative functionalities are currently offered. This chapter is concluded, in Section 2.4, by identifying the main state-of-the-art bottlenecks in the specification of a semantic multimedia remote viewer for collaborative mobile thin clients.

By advancing a novel architectural framework, *Chapter 3* offers a solution in this respect. The principle is presented in Section 3.1, while the details concerning the architectural design are presented in Section 3.2. In this respect, the main novel blocks are: XGraphic Listener, XParser, Semantic MPEG-4 Converter, Semantic Controller, Pruner, Semantic Adapter, Interaction Enabler, Collaboration Enrichment, Collaboration and Interaction Event Manager and Collaboration and Interaction Handler.

Chapter 3 is devoted to the evaluation of this solution, which was benchmarked against its state-of-the-art competitors provided by VNC (RFB) and Microsoft (RDP). It was demonstrated that: (1) it features high level visual quality, *e.g.* PSNR values between 30 and 42dB or SSIM values larger than 0.9999; (2) the downlink band-width gain factors range from 2 to 60 while the up-link bandwidth gain factors range from 3 to 10; (3) the network roundtrip-time is reduce by factors of 4 to 6; (4) the CPU activity is larger than in the Microsoft RDP case but is reduced by a factor of 1.5 with respect to the VNC RFB.

The *Chapter 4* concludes the thesis and open perspectives for future work.

The references and a list of abbreviations are also presented.

The manuscript contains three *Appendixes*. The first two present the detailed descriptions of the underlying patents, while *Appendix III* gives the conversion dictionary used for converting the XProtocol requests into their MPEG-4 BiFS and LASeR counterparts.

Chapter 2. State of the art

Ce chapitre donne un aperçu critique sur les solutions existantes pour instancier les systèmes de rendu distant sur les terminaux mobiles légers (X, VNC, NX, RDP, ...). Cette confrontation entre les limites actuelles et les défis scientifiques / applicatives met en exergue que : (1) une vrai expérience multimédia collaborative ne peut pas être offerte au niveau du terminal, (2) la compression du contenu multimédia est abordée d'un seul point de vue image statique, ainsi entraînant une surconsommation des ressources réseau; (3) l'inexistence d'une solution générale, indépendante par rapport aux particularités logicielles et matérielles du terminal, ce qui représente un frein au déploiement des solutions normatives.

Par conséquent, définir un système de rendu distant multimédia pour les terminaux légers et mobiles reste un fort enjeu scientifique avec multiples retombées applicatives. Tout d'abord, une expérience multimédia collaborative doit être fournie côté terminal. Ensuite, les contraintes liées au réseau (bande passante, erreurs et latence variantes en temps) et au terminal (ressources de calcul et de mémoire réduites) doivent être respectées. Finalement, l'acceptation par le marché d'une telle solution est jalonnée par son indépendance par rapport aux producteurs de terminaux et par le soutien offert par les communautés.

2.1 Content representation technologies

2.1.1. Introduction

Any thin client solution can be accommodated by a client-server model, where the client is connected to the server through a connection managed by a given protocol. From the functional point of view, the software application (text editing, www browsing, multimedia entertainment, …) runs on the server and generates some semantically structured graphical output (*i.e.* a collection of structured text, image/video, 2D/3D graphics, …), see Figure 2.1. This graphical content should be, in the ideal case, transmitted and visualized at the client-side.

Consequently, a key issue in designing a thin client solution is the choice of multimedia representation technology. Moreover, in order to fully reproduce the same experience on the user's terminal, it is not sufficient to transmit only the raw audio-visual data. Additional information, describing the spatio-temporal relations between theses elementary data should be also available.

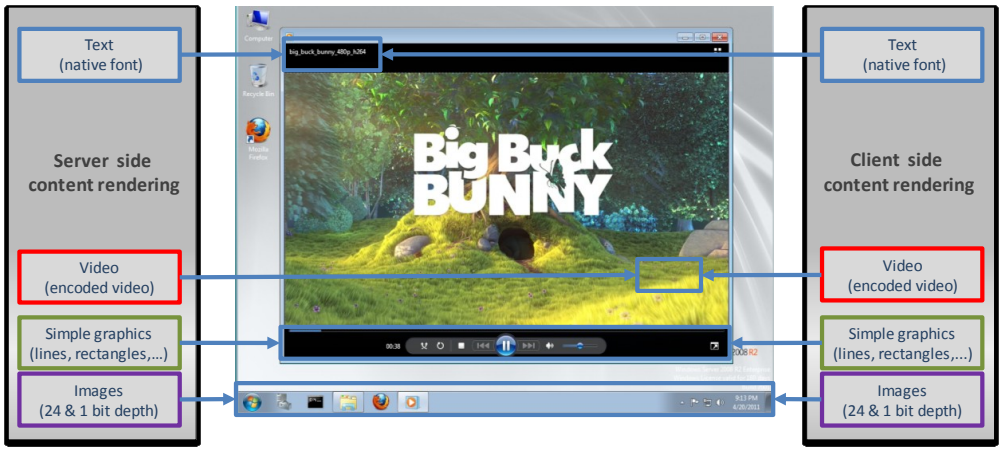


Figure 2.1. Scene technology support for mobile thin clients

This basic observation brings to light the potentiality of multimedia scenes for serving the thin client solutions. A multimedia scene is composed by its elementary components (text, image/video, 2D/3D graphics, …) and by the spatio-temporal relations among them (these relations are further referred to as scene description). Actually, scene description specifies four aspects of a multimedia presentation:

- how the scene elements (media or graphics) are organized spatially, *e.g.* the spatial layout of the visual elements;
- how the scene elements (media or graphics) are organized temporally, *i.e.* if and how they are synchronized, when they start or end;

- how to interact with the elements in the scene (media or graphics), *e.g.* when a user clicks on an image;
- how the scene changes during the time, *e.g.* when an image changes its coordinates.

The action of transforming a multimedia scene from a common representation space to a specific presentation device (*i.e.* speakers and/or a multimedia player) is called *rendering*.

By enabling all the multimedia scene elements to be encoded independently the development of authoring, editing, and interaction tools are alleviated. This permits the modification of the scene description without having to decode or process in any way the audio-visual media.

2.1.2. Comparison of content representation technologies

Amongst the technologies for heterogeneous content representation existing today, we will consider the most exploited by the mobile thin client environment: BiFS [BiFS, 2005], LASeR [LASeR, 2008], Adobe Flash [Adobe, 2005], Java [Java, 2005], SMIL/SVG [SMIL/SVG, 2011], [TimedText, 2010], [xHTML, 2009].

We benchmarked all the solutions according to their performances in the areas of binary encoding, dynamic updates and streaming.

Binary encoding

Multimedia scene binary encoding is already presented by several market solutions: BiFS, LASeR, Flash, Java..., to mention but a few. On the one hand, LASeR is the only technology specifically developed addressing the needs of mobile thin devices requiring at the same time strong compression and low complexity of decoding. On the other hand, BiFS takes the lead over LASeR by its power of expression and its strong graphics features with their possibility for describing 3D scenes.

A particular case is represented by the xHTML technology which has no dedicated compression mechanism, but exploits some generic lossless compression algorithms (*e.g.* gzip) [Liu, 2005], [HTTP, 1999].

Dynamic updates

Dynamic updates allow the server to modify the multimedia scene in a reactive, smooth and continuous way [Song, 2011]. In this respect, commands permitting scene modifications (object deletion / creation / replacement) in a timely manner [Song, 2011] should be provided inside the considered technology. This is the case of BiFS, LASeR and Flash. xHTML does not directly allow dynamic updates, but delegates this responsibility to additional technology (*e.g.* JavaScript) [JavaScript, 2011].

Streaming

Streaming refers to the concept of consistently transmitting and presenting media to an end user at a rate determined by the media updating mechanism *per se*; *live streaming* refers to the instantaneous transmission of some media created by a live source. BiFS and LASeR are the only binary compressed content representations intrinsically designed to be streamed. In this respect, dedicated mechanisms for individual media encapsulation into a binary format have been standardized and generic transmission protocols are subsequently employed for the corresponding streams. Note that the Flash philosophy does not directly support such a distribution mode: the *swf* file is generated on the server and then downloaded to the client which cannot change its functionalities. However, inside the *swf* file, Flash does provide tools for streaming external multimedia contents with their own native support, *e.g.* a FLV video can be streamed inside the Flash player. A similar approach is followed by xHTML.

Conclusion

The current solutions can be represented in terms of power of expression and graphic features, see Figure 2.2. This figure was obtained by extending a similar representation in [LASeR, 2006]. The power of expression (on the abscissa) represents the possibility of describing complex/heterogeneous scenes. The graphics features (on the ordinate) relates to the visual quality of the displayed content.

It can be seen that LASeR is *a priori* the most suitable technology for creating mobile thin applications. BiFS is the second best solution, with more powerful tools for describing complex heterogeneous scenes, with high quality elementary components. These two technologies will be detailed below.

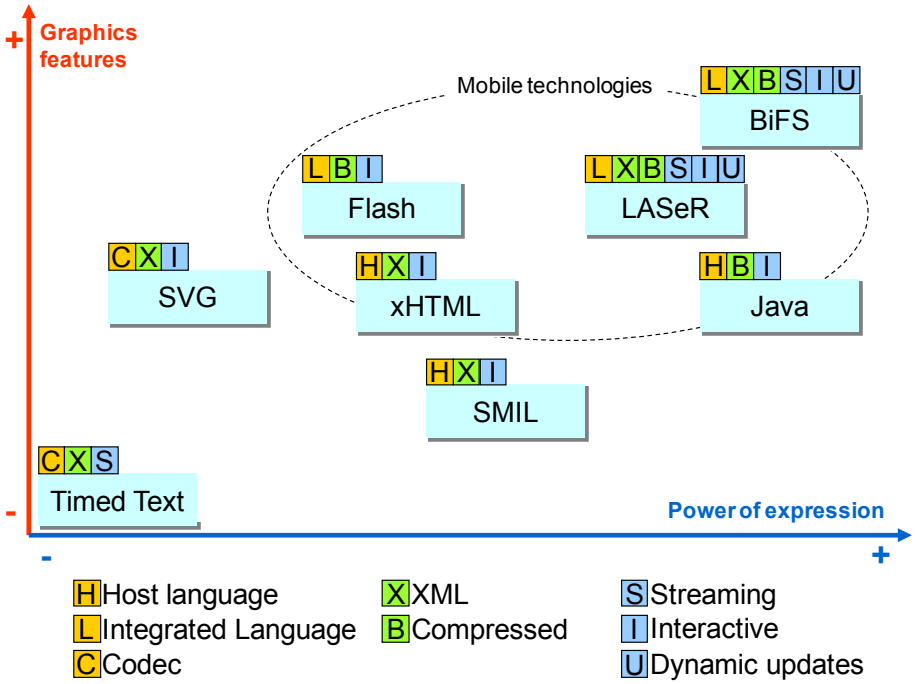


Figure 2.2. Concurrent solutions for heterogeneous content encoding, updating and streaming

2.1.3. BiFS and LASeR principles

Overview

We will investigate the existing multimedia scene technologies and we will discuss the peculiarities of BiFS and LASeR as well as their potential for serving mobile remote display purposes.

The MPEG-4 audio-visual scenes are composed of diversity media objects, structured in a hierarchical order forming a tree. At the ends of the hierarchy, two types of objects can be generally found: multimedia objects and primitive media objects. While analyzing one heterogeneous scene, Figure 2.1, we can notice the following multimedia objects:

- images (*e.g.* uncompressed RAW, or compressed png and/or jpeg, …);
- video objects (*e.g.* real-time video stream);
- audio objects (*e.g.* the audio from the video streamed);

followed by the primitive media objects, capable of representing synthetic content:

- text (*e.g.* representation of an textual information);
- graphics (*e.g.* lines, rectangles, …).

Such an object partitioning allows the content creators to construct complex scenes and enables the users to interact and manipulate them.

Binary Format for Scenes (BiFS)

MPEG-4 defines a dedicated description language, called Binary Format for Scene (BiFS) [Battista, 1999], [Battista, 2000], which is able to describe the heterogeneous content of the scene, to manage the scene object behavior (*e.g.* object animation) and to ensure the timed and conditional updates (*e.g.* user input/interactivity). While BiFS at the content representation level BiFS can be considered as an additional layer over VRML, it also provides supports for optimized content compression and delivery.

A BIFS scene is represented as a hierarchical structured (a *tree*) of *nodes*[2]. Each node contains not only information about the audio-visual object in the scene but also about the spatio-temporal relations among such objects (*i.e.* the scene description), about the user possibility to interact with that object, *etc*. Individual nodes can be logically grouped together, by using a devoted node (the grouping node), see Figure 2.3. Note that the scene description can evolve over time by using scene description updates.

The novelty of BiFS does not only relate to the scene description but also to the scene compression. Traditionally, the heterogeneous visual content to be remotely displayed was represented by successive frames composing a single video to be eventually compressed by some known codec (such as MPEG-2 [MPEG-2, 2007] or MPEG-4 AVC [AVC, 2012]). BiFS follows a completely different approach, by allowing each object to be encoded with its own coding scheme (video is coded as video, text as text, and graphics as graphics).

In order to facilitate the user interaction with the audio-visual representation, BiFS supports interaction between the user and the objects. The interactivity mechanisms are integrated within the scene description information referred to as *sensors*, which are special nodes that can trigger user events based on specific conditions (*e.g.* keyboard key pressed and/or mouse movements). These sensors can handle two types of interactivity: client-side and server-side.

The client-side interactivity deals with content manipulation on the end user terminal, where only local scene updates are available: the user events are captured and the scene description is correspondingly updated, without contacting the server.

[2] Although the scene elements are structured in a tree, the standard name is the *scene-graph*.

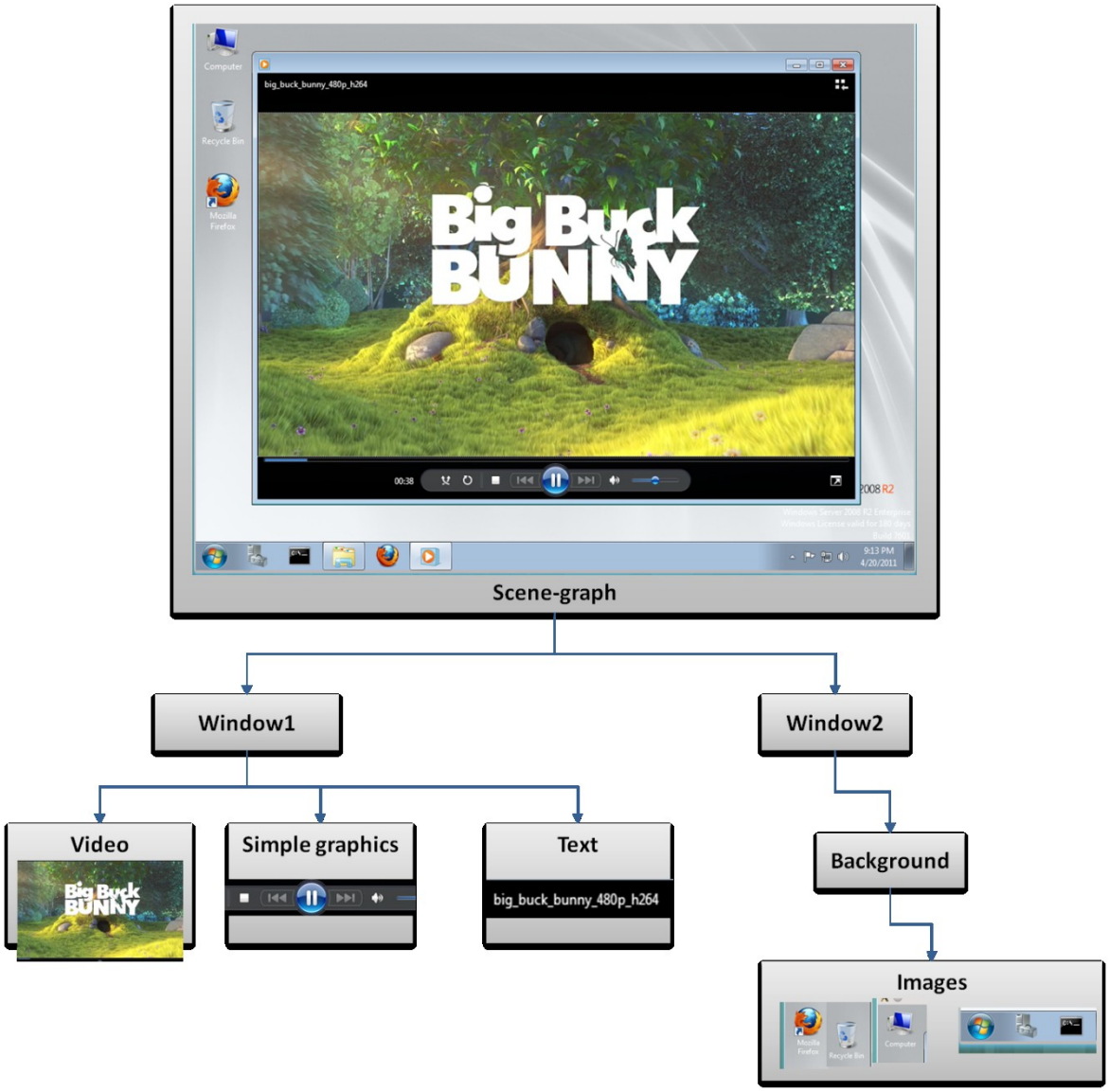


Figure 2.3. BiFS scene-graph description example

The server-side interactivity supposes that the user events are sent to the server by using an up-link channel. MPEG-4 provides two possible solutions for ensuring the server-side interactivity. First, the ECMA script (JavaScript language) can be considered in order to enable programmatic access to MPEG-4 objects. In order to achieve server-side interactivity, an *AJAX HttpRequest* [Bruno, 2006] object is used to send user interactivity information to the server. In the particular case of BiFS, a second interactivity mechanism is provided by the *ServerCommand* [BiFS, 2006] which allows the occurrence of a user event to be directly signaled from the scene to the server.

An example of a BiFS scene description, considering the Figure.2.3 represented in a textual VRML format is represented in Code 2.1, follows:

```
InitialObjectDescriptor {
 objectDescriptorID 1
 audioProfileLevelIndication 255
 visualProfileLevelIndication 255
 sceneProfileLevelIndication 254
 graphicsProfileLevelIndication 254
 ODProfileLevelIndication 255
 esDescr [
  ES Descriptor {
   ES_ID 1
   decConfigDescr DecoderConfigDescriptor {
    streamType 3
    decSpecificInfo BIFSConfig {
     isCommandStream true
     pixelMetric true
     pixelWidth 800
     pixelHeight 600
    }
   }
  }
 ]
}
orderedgroup DEF Scene-graph OrderedGroup {
 children [
  DEF B Background2D {
   backColor 1 0 0
  }
  WorldInfo {
   info [
    "Example"
   ]
   title "examplifying the Figure 4.3"
  }
  transofrm DEF Window1 Transform2D {
  children [
    transform DEV Video Transform2D {
      scale 1 1
      children [
        Inline {
        url [OD:8]
        }
      ]
    }
    transform DEF SimpleGraphics Transform2D {
      children [
        Shape {
          appearance Appearance {
              material Material2D {
             emissiveColor 0 0 0
             filled TRUE
            }
          }
          geometry Rectangle {

          }
        }
        Shape {
          appearance Appearance {
              material Material2D {
             emissiveColor 0 0 0
             filled TRUE
            }
          }
          geometry Line {

          }

          .......
        }
      ]
    }
    transform DEF Text Transform2D {
      children [
```

```
        Shape {
          appearance DEF TA Appearance {
            material Material2D {
              emissiveColor 1 1 1
              filled TRUE
            }
          }
          geometry Text {
            string [ "big buck bunny 480p h264" ]
            fontStyle DEF FS FontStyle {
              size 20
              family "SANS"
            }
          }

        }
      ]
         }

    ]
  }
  transofrm DEF Window2 Transform2D {
    children [
      transform DEF Background Transform2D {
        children [
          Shape {
            appearance Appearance {
              material Material2D{
                emissiveColor 1 0 1
                filled TRUE
              }
            }
            geometry Rectangle {
              radius 50
            }
          }
          transform DEF Images Transform2D {
            translation 80 90
            children [
              Shape {
                appearance Appearance {
                  texture PixelTexture {
                    image 4 4 2
                    0x00FF 0x00FF 0x0000 0x0000
                  }
                }
                geometry Circle {
                  radius 50
                }
              }
              Shape {
                appearance Appearance {
                  texture PixelTexture {
                    image 4 4 2
                    0x00FF 0x00FF 0x0000 0x0000
                  }
                }
                geometry Circle {
                  radius 50
                }
              }
              ........
            ]
          }
        ]
      }
    ]
  }
  ]
}

AT 0 {
  UPDATE OD [
```

```
    ObjectDescriptor {
     objectDescriptorID 8
     URLstring "big buck bunny.mp4"
    }
   ]
}
```

Code 2.1. Example of BiFS scene-graph, represented using VRML

The complete BiFS scene description, corresponding to Figure 2.3 has the following structure:

- a header that contains some global information about the encoding;
- a binary value representing the Transform node;
- a bit specifying that the fields of the Transform node will be specified by their index, rather than in an exhaustive list;
- the index for the 'translation' field;
- a binary encoding of the SFVec2f value 0 0 (since there is no quantization defined here, this encoding consists of three 32-bit values; during decoding, the decoder knows the type of the field it is reading and thus knows how many bits to read and how to interpret them);
- the index of the children field of the Transform node;
- the binary representation of the Shape node, which is:
    - a binary value for the Shape node;
    - a bit specifying that all of the fields of the Shape node and their values will be listed sequentially rather than by index/value pairs;
    - a binary representation for the Rectangle node which is:
        - a binary value for the Rectangle node;
        - a bit specifying that the fields of the Cube will be specified by index;
        - the index of the 'size' field;
        - a binary encoding of the SFVec2f value 1 1;
        - a bit specifying that no more fields for the Rectangle node will be sent;
    - a binary value for the Appearance node, followed by its encoding(omitted here);
- a bit terminating the list of fields for the Transform node.

Lightweight Application Scene Representation (LASeR)

The BiFS principles have been further optimized for thin clients and mobile network purposes, thus resulting in a standard called Lightweight Application Scene Representation (LASeR) [LASeR, 2005], [Dufourd, 2005].

Properly referred to as MPEG-4 Part 20, MPEG-4 LASeR is designed for representing and delivering rich-media services to resource-constrained devices such as mobile phones. A LASeR engine, Figure 2.4, has rich media composition capabilities relying on the usage of an SVG scene tree. After binary encoding of the LASeR scene, the LASeR commands are the main enablers for

dynamic scene updating and real-time streaming. The LASeR binary format is based on a generic Binary MPEG (BiM) [BiM, 2006] format, which applies encoding according to an already known XML schema. This approach makes the BiM format a schema aware encoding, *i.e.* it is based on the mutual knowledge of the schema between both the server (encoder) and the client (decoder).

As previously stated LASeR is capable of capturing the user events at the scene description level.

When considering the high demands of interactive LASeR services, multiple connections from different audio-visual media objects, distributed on different locations, should be supported, hence a new type of service is required. In this respect, the Simple Aggregation Format (SAF) is specified so as to enable the creation of a single LASeR stream in an efficient way, ready to be streamed through the network.

The general overview of the LASeR brings to light that:

- it is devoted only to 2D scenes encoding, including vector graphics, and timed modifications of the scene;
- SAF (Simple Aggregation Format) alleviates the aggregation of all the streams into a single LASeR stream.

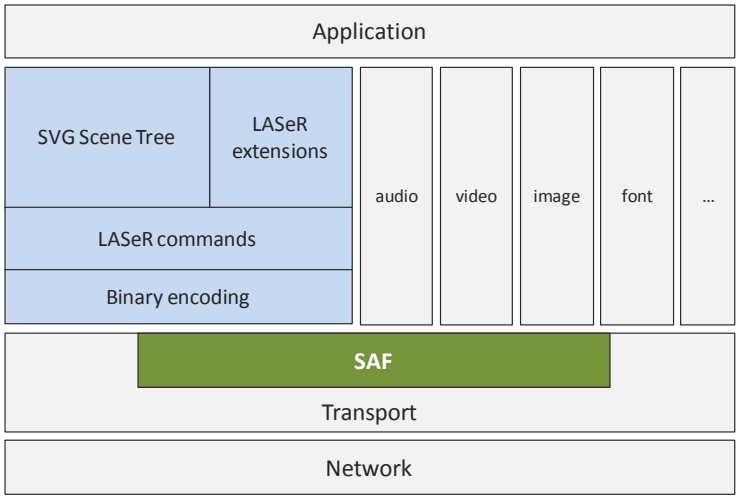


Figure 2.4. LASeR architecture (*cf.* [LASeR, 2005])

```
<?xml version="1.0" encoding="UTF-8"?>
  <saf:SAFSession xmlns:saf="urn:mpeg:mpeg4:SAF:2005" …>
    <saf:sceneHeader>
      <LASeRHeader …/>
    </saf:sceneHeader>

  <saf:RemoteStreamHeader streamID="Video0" objectTypeIndication="32" streamType="4"
source="[video stream]"/>
  <saf:endOfStream ref="Video0"/>

      <saf:sceneUnit>
        <lsru:NewScene>
          <svg width="800" height="600" viewBox="0 0 800 600"
version="1.1"baseProfile="tiny">
     <g lsr:id="Window1" lsr:translation="45 0">
```

```
        <video begin="2" xlink:href="#Video0" width="100" height="80" repeatCount="3"
transformBehavior="pinned"/>
        <rect  transform="translate(165, 220)" fill="white" stroke="white"></rect>
        <line  transform="translate(165, 220)" fill="white" stroke="white"></line>
        <text font-family="SYSTEM" font-size="12" font-style="italic" id="text" y="88"
text-anchor="start" display-align="before">
          Un text
        </text>
      </g>
      <g lsr:id="Window2" lsr:translation="45 0">
        <rect  transform="translate(165, 220)" fill="gray" stroke="red"></rect>
        <image id="image1JPEG" x="" y="0" width="" height="150"
xlink:href="../icon1.png"/>
        <image id="image1JPEG" x="" y="0" width="" height="150" xlink:href="..."/>
        ....
      </g>
            </svg>
         </lsru:NewScene>
       </saf:sceneUnit>
  <saf:endOfSAFS>
```

Code 2.2. LASeR scene example of the Figure 4.3, including SAF aggregation

2.1.4. Conclusion

MPEG-4 BiFS and LASeR are potentially capable of fulfilling key remote display requirements:

- the heterogeneous content generated by the application can be aggregated into a multimedia MPEG-4 scene-graph, and the related semantic information can be used for the management of this graph;
- the compression of each type of content (text, audio, image, graphics, video, 3D) by dedicated codecs and the related live streaming are possible by using the corresponding BiFS/LASeR technologies;
- the user interactivity can be established both locally and remotely;
- the client CPU activity may concern only light-weight operations (scene-graph rendering and basic user event handling) while the computational intensive operations (scene-graph creation/management and user event management) may be performed by the server.

Besides these technical properties, BiFS and LASeR also have the advantage of being stable, open international standards, reinforced by open source reference software supports.

2.2 Mobile Thin Clients technologies

2.2.1. Overview

Nowadays, all the thin clients solutions (be they wired or wireless, desktop computer or thin client oriented, Windows or Unix based, *etc.*) exploit the client-server architecture. Consequently, any remote display technological support can be assessed according to the following three criteria: (1) the level of interception of the visual content, generated by the application at the server side, (2) the compression methods and the protocol used for transmission of the content to the client, and (3) the management of the user interactivity (including the transmission of the user events from client to server). When targeting mobile thin clients, an additional fourth criterion related to the energy consumption is taken into account. The study in [Carroll, 2010] brought to light that the energy consumption on a smartphone depends on the network (GSM/Wi-Fi), CPU, RAM, display and audio. While the last three factors are rather related to the device and to the user behavior, the amount of data transmitted through the network and the CPU activity intrinsically depend on the technology and will be further investigated in our study.

The present section considers the most often encountered desktop thin clients support technologies (X window, NX, VNC, and RDP) and discusses them according to these criteria.

2.2.2. X window system

The **X window system** represents native thin client framework for all current day desktops, and it is exploited mostly by Linux applications accommodating an XClient and an XServer connected trough XProtocol. The X window system terminology defines the user terminal, where the applications are displayed as the XServer, and the server running the application as the XClient [Nye, 1990]. Based on its specification and implementation, the client and the server are able to run on the same machine (PC) or distributed, by using several hardware architectures and operating systems (Unix, Linux, …), see Figure 2.5.

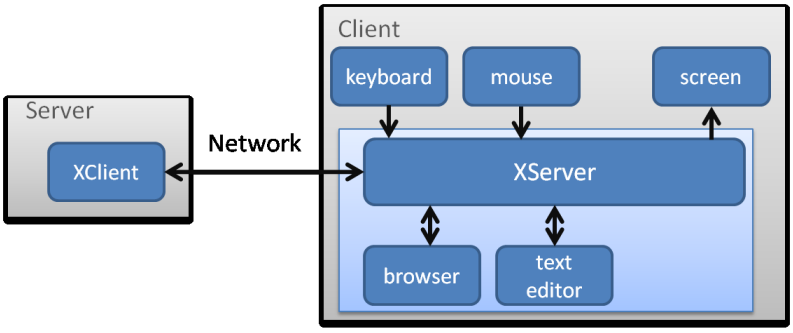


Figure 2.5. X windows system server-client architecture

The graphical output generated by the application Graphical User Interface (GUI) is traditionally structured in a hierarchical order, defining a top level element, usually a *Window,* and followed by other windows or elements as children to the root window. The communication protocol (XProtocol) between the server and the client was design to support a basic set of 119 requests, generated by the application output. This protocol ensures all bi-directional communication tasks but makes no provision for content compression. Besides the requests, the XProtocol structure has replies, events, and errors:

- *request*: the client requests information from the server or requests an action (like drawing, menu closing, …);
- *reply:* the server responds to a request (not all requests generate replies);
- *event:* the server sends an event to the client (*e.g.* keyboard or mouse input, or a window being moved, resized or exposed);
- *error:* the server sends an error packet if a request is invalid.

Although particular applications may require some graphic extensions, the practice shows that a sub-set of 20 graphical requests are sufficient for displaying the large majority of application. As an example, when considering www browsing for 5 minutes, more than 70% of the total number of generated graphic primitives is covered by: *CreatePixmap, PutImage, CopyArea, CreateGC*, *PolyFillRectangle*, *PolyRectangle*, *PolySegment*, *FillPoly*, *PolyLine*, *CreateWindow, ConfigureWindow, PolyText8*.

The rendering mechanism for the X windows system is illustrated in Figure 2.6.

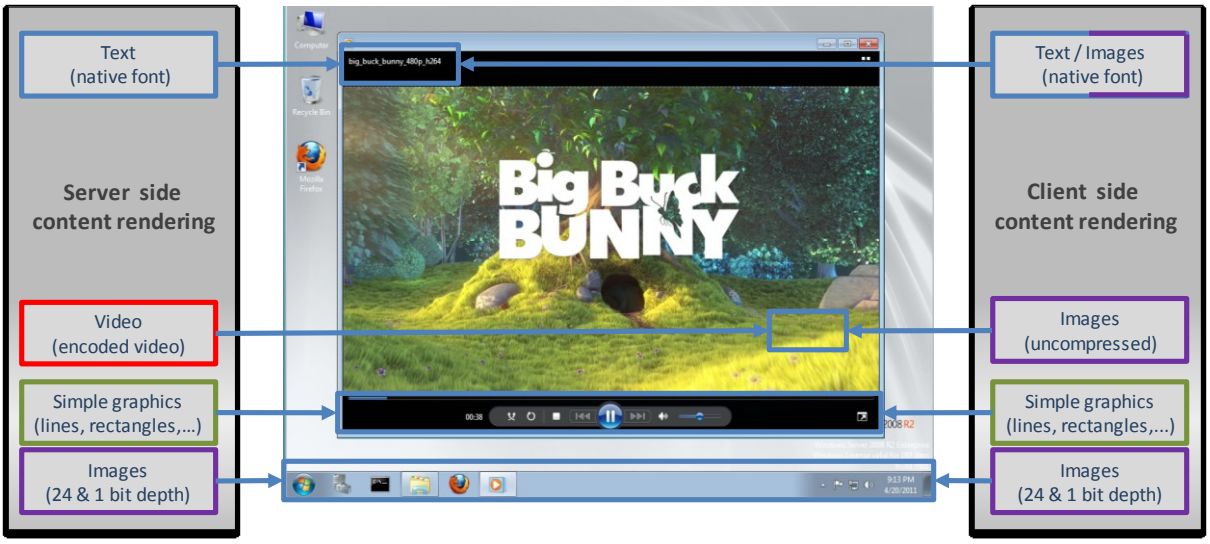


Figure 2.6. X window system server and client side content rendering

An example of X polySegment, putImage and polyText16, graphical requests, uncompressed description (including the used bites) follows:

```
X polySegment

Bytes      type      description
1          66    opcode
1       unused
2          3+2n      requestlength
4          DRAWABLE   drawable
4          GCONTEXT   gc
8n         LISTofSEGMENT    segments
```

Code 2.3. X Protocol request description, polySegment

```
X putImage

Bytes      type      description
1          72      opcode
1              format
0          Bitmap
1          XYPixmap
2          ZPixmap
2          6+(n+p)/4    requestlength
4          DRAWABLE   drawable
4          GCONTEXT   gc
2          CARD16   width
2          CARD16   height
2           INT16   dst-x
2          INT16      dst-y
1          CARD8    left-pad
1          CARD8    depth
2              unused
n          LISTofBYTE     data
p              unused, p=pad(n)
```

Code 2.4. X Protocol request description, putImage

```
X polyText16

Bytes      type      description
1          74      opcode
1              unused
2          4+(n+p)/4    requestlength
4          DRAWABLE   drawable
4          GCONTEXT   gc
2          INT16    x
2          INT16    y
n          LISTofTEXTITEM8  items
p          unused, p=pad(n) (p is always 0 or 1)
```

Code 2.5. X Protocol request description, polyText16

While ensuring good performances when implemented on a single desktop environment, the X window system cannot be directly employed in distributed environments (where the client and the server installed on separate machines). For instance, the video generated at the server side cannot be displayed as a video *per-se* at the client side. By default, the video is converted in RAW (uncompressed) sequences of images which are subsequently transmitted and displayed at the client side, see Figure 2.6. The same situation may occur for other type of content, like the

fonts. Consequently, an artificially overcharged traffic is generated between server and client, thus making it impossible for the X window system to be implemented for mobile thin client applications.

On the client side, the XServer only displays the graphical content without making a provision for capturing the user interactivity. The user events are captured at the XClient only by generic Linux/Unix OS mechanisms (keyboard/mouse drivers).

By summarizing the X window system functionalities, we can notice the following peculiarities:

- XServer
  - displaying the graphical requests.
- XClient
  - executes the applications;
  - captures the user interaction.
- XProtocol
  - each attribute from the specification has a fixed length;
  - no provision for compression;
  - no provision for video streaming in distributed environments.

2.2.3. NoMachine NX technology

By providing an alternative protocol, NoMachine propose an open solution, **NX technology**, intended to reduce the X Protocol network consumption and latency, while benefiting from the complete X windows system functionality. In this respect, two new modules, the NX Agent and NX Proxy, are designed, see Figure 2.7.

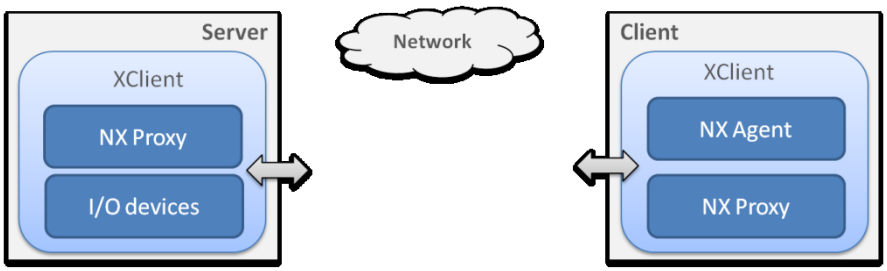


Figure 2.7. NoMachine architecture

The NX Proxy is responsible for applying a compression and decompression to the XProtocol. Hence, its implementation is required at both the XServer and XClient sides and an underlying protocol (the NX protocol) is defined accordingly.

The NX Agent is required only at the client side, in order to avoid the unnecessary XProcotol data round trips.

The NX technology considers all XProtocol message to be composed of two parts: a fixed size part called identity, and a dynamic size part called data. This way, the NX compression algorithm can be applied to the data, where the information is dynamically generated and likely to be different on each message. For instance, consider the case of the compression of the polySegment XProtocol request. The data part for the polySegment is a list of 2D coordinates, each of which is represented as signed integers. The NX compression is achieved by representing each coordinate in the list as a relative value with respect to its predecessor. This way, on average, an X polySegment request of 32 bytes can be fully encoded in 32 bits (an average compression ratios ranging of 8:1). This compression mechanism is illustrated below, as the C language representation of the X Protocol polySegment request from Xproto.h library:

```
#define X_PolySegment        66
#define sz_xPolySegmentReq   12
typedef struct {
    CARD8                    reqType;
    BYTE            pad;
    CARD16 length   B16;
    Drawable drawable B32;
    GContext gc     B32;
} xPolySegmentReq;
```

Code 2.6. Code sample written in C language for compressing the polySegment X request

However, the basic X windows limitations in video/text representation in distributed environments are still present, see Figure 2.8: this type of content is converted into images which are displayed on the client side. However, the network consumption is alleviated by introducing a special NX Protocol message, *NX_PutPackedImage* allowing the transmission of compressed images, in the JPEG or PNG formatted [JPEG, 1996], [PNG, 2004], for instance.

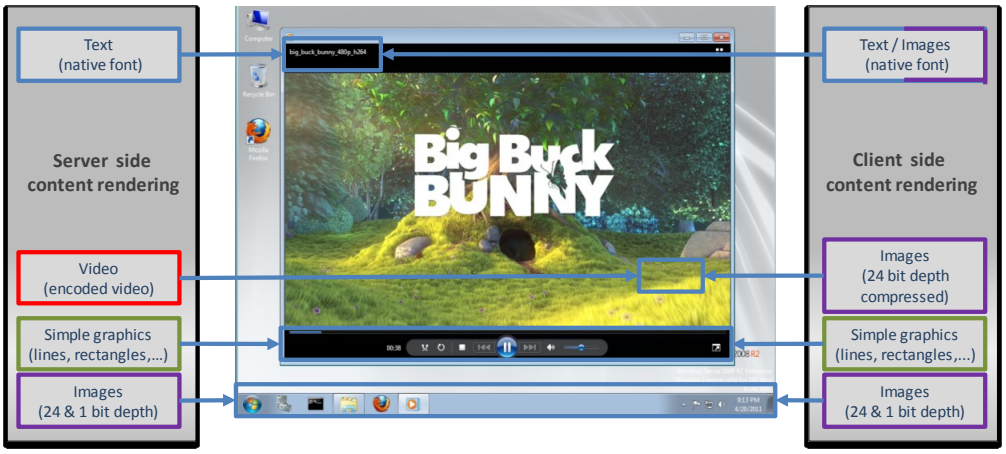


Figure 2.8. NX client-server rendering

NX also inherits the X windows limitations in terms of user interactivity (managed at the OS driver level) and CPU consumption at the client side (even increased by the need for the NX Proxy and NX Agent to be accommodated).

To conclude with, although the experiments showed a very good compression rate of the initial X content [Yang, 2002], such a solution is not yet available for real-life mobile thin client applications.

2.2.4. Virtual Network Computing

The **VNC** (**Virtual Network Computing**) is a thin client solution developed totally independent with respect to the operating system. It was firstly deployed on X window system (both XClient and XServer) bringing new software components in order to jointly alleviate the bandwidth and CPU constraints.

It is based on the Remote FrameBuffer (RFB) protocol [Richardson, 2011]. At the server side, it assumes that all the graphical content generated by the application is already converted into a sequence of RAW images, stored into a frame buffer, see Figure 2.9. The content of these frames is analyzed (by some image processing techniques) and according to its type and to the client capability, a compression algorithm is applied so as to obtain the targeted image quality (*e.g.* 1 pixel-depth images for representing the text, see Figure 2.10). At the client side, the images are simply decompressed and rendered.

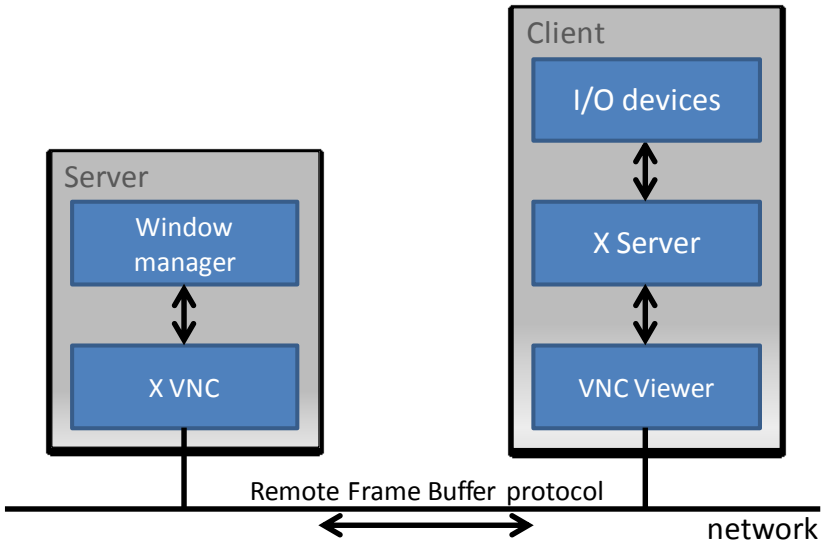


Figure 2.9. The VNC server-client architecture

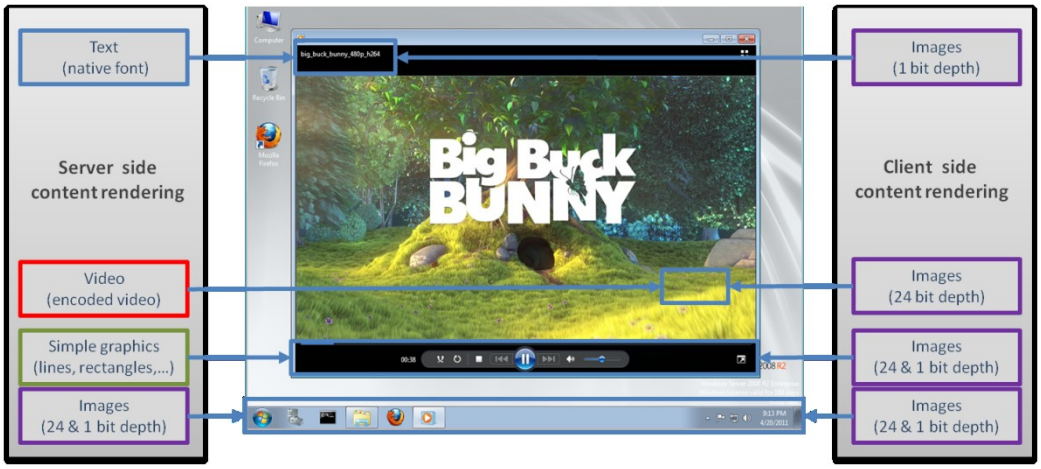


Figure 2.10. VNC server-client content rendering

As it can be seen, the VNC architecture considers only pixels, giving the flexibility to the developers to use/create different image compression algorithms. The only requirement by the VNC thin client is that all the client, server and protocol modules, must support the basic RAW image. The display side of the protocol is base on a single graphic primitive:

```
Put a rectangle of pixel data at a given x, y position
```

Code 2.7. Graphical primitive used by VNC server

As supported by the open-source community, several image compression optimizations are currently considered: TightVNC, TurboVNC, VNC-HEXTILE, VNC-ZRLE, … . For current day mobile thin clients (limited client CPU activity and bandwidth consumption), VNC-HEXTILE can be considered as optimal and effective compression method [Richardson, 2011].

The VNC HEXTILE compression is based on rectangles that are split up into 16x16 tiles, to be sent in a predetermined order.

The sample code from a C function, part of the VNC HEXTILE encoding, follows:

```
static void hextile enc cord(uint8 t *ptr, int x, int y, int w, int h)
{
    ptr[0] = ((x & 0x0F) << 4) | (y & 0x0F);
    ptr[1] = (((w - 1) & 0x0F) << 4) | ((h - 1) & 0x0F);
}
```

Code 2.8. VNC HEXTILE encoding function expressed in C language

The areas with individual pixel color refers to a pixel format, which can be 24-bit or 16-bit "true color", are translated directly into red, green, and blue intensities.

```
PIXEL_FORMAT

No. of bytes                Type [Value]       Description
1                  CARD8              bits-per-pixel
1                  CARD8              depth
1                  CARD8              big-endian-flag
1                  CARD8              true-colour-flag
2                  CARD16             red-max
2                  CARD16             green-max
2                  CARD16             blue-max
1                  CARD8              red-shift
1                  CARD8              green-shift
1                  CARD8              blue-shift
3                                     padding
```

Code 2.9. VNC PIXEL FORMAT encoding function expressed in C language

RFB ensures simple user events to be sent from the server whenever the user presses a key or pointing device button, or whenever the pointing device is moved. Moreover, it allows user events from other non-standard I/O devices (*e.g.* a pen-based handwriting recognition engine might generate keyboard events).

To conclude with, according to the RFB protocol, all the heterogeneous content generated by the application output (text, simple graphics, video) are represented by pixel data, see Figure 2.10, thus referring the user from a full multimedia experience.

2.2.5. Microsoft RDP

Microsoft Windows OS provides the **RDP (Remote Display Protocol) framework**, a proprietary protocol solution for thin clients, available on the market in both desktop and mobile versions. It is natively included into almost each Windows OS (Windows 2000 Server, Windows XP, Windows 7, *etc.*), allowing windows clients to connect to the server without additional modules installation. From the specification point of view, RDP is closely connected to the content generated by the GDI (Graphical Device Interface), which is the operating system's visual display, see Figure 2.11.

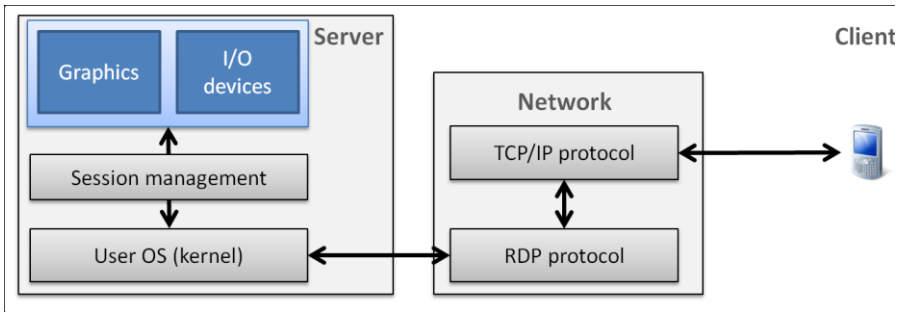


Figure 2.11. Windows RDP server-client architecture

At the server side, the RDP uses the video output driver to intercept the application output, parses the GDI data and usually represents it by a mixture of images, graphics and formatted text. This content is then transmitted through the RDP protocol (based on a TCP/IP connection type) to the client where it is processed and the corresponding GDI is used for displaying the received graphic content, see Figure 2.12.

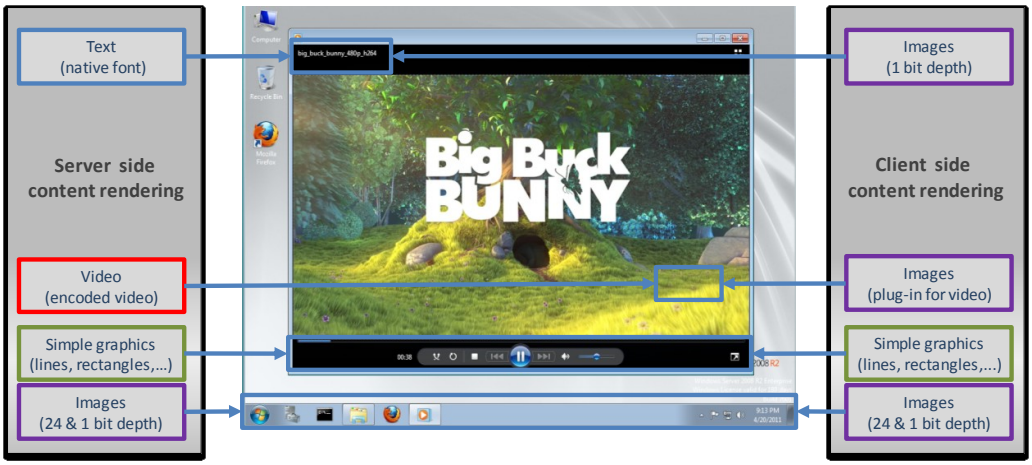


Figure 2.12. Microsoft RDP server-client content rendering

The simple example of the RDP rectangle message (TS_RECTANGLE16), which is a rectangle displayed within inclusive coordinates (*i.e.* the rectangle is defined by right/bottom and left/up corners) follows.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
left																top															
right																bottom															

```
left (2 bytes):  A 16-bit, unsigned integer - the  leftmost bound of the rectangle.
top (2 bytes):   A 16-bit, unsigned integer - the upper bound of the rectangle.
right (2 bytes): A 16-bit, unsigned integer - the  rightmost bound of the rectangle.
bottom (2 bytes):  A 16-bit, unsigned integer - the lower bound of the rectangle.
```

Code 2.10. Binary description of an RDP rectangle message

When displaying a pixel, the *RDPGFX_PIXELFORMAT* is specifying the color component layout in a pixel format having the following description:

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	20	1	2	3	4	5	6	7	8	9	30	1
format																															

```
format (1 byte): An 8-bit unsigned integer that specifies the pixel format.

Value        Meaning
PIXEL_FORMAT_XRGB_8888 0x20    32bpp with no valid alpha (XRGB)
PIXEL_FORMAT_ARGB_8888 0x21    32bpp with valid alpha (ARGB)
```

Code 2.11. Binary description of an RDP image pixel message

For fixed desktop environments, the RemoteFX, an emerging extension of the basic RDP framework, is the main enabler for full multimedia content transmission [RemoteFX, 2011].

The user interactivity is managed by the RDP and/or Windows OS drivers.

2.2.6. Conclusion

The performances exhibited by the remote display technologies presented in Section 2.2 are synoptically illustrated in Figure 2.13. It can be noticed that these technologies feature no direct support for multimedia (except for the RDP RemoteFX in desktop environments), none of them is compatible with the ISO multimedia standards and several requirements are still to be met when designing a mobile thin client remote display:

- interception of visual content: capturing the graphical content at the lowest possible levels (thus ensuring generality) while retaining the semantic information of the content (thus preserving the content type and providing multimedia experience);
- visual content compression/transmission: deploying an efficient compression algorithm for the handling of heterogeneous content;
- user interactivity: ensuring a prescribed QoE (Quality of Experience) for the user interactivity, irrespective of the application (text editing, www browsing, entertainment, …), of the network bandwidth (both up-link and down-link) and of the type of terminal;
- CPU activity: specifying low-complexity algorithms, coping with the CPU limits imposed by the thin clients.

The present thesis considers the possibility of using the MPEG-4 technologies for multimedia scenes to jointly solve these four issues (see the *Targeted solution* in the low-left area in Figure 2.13).

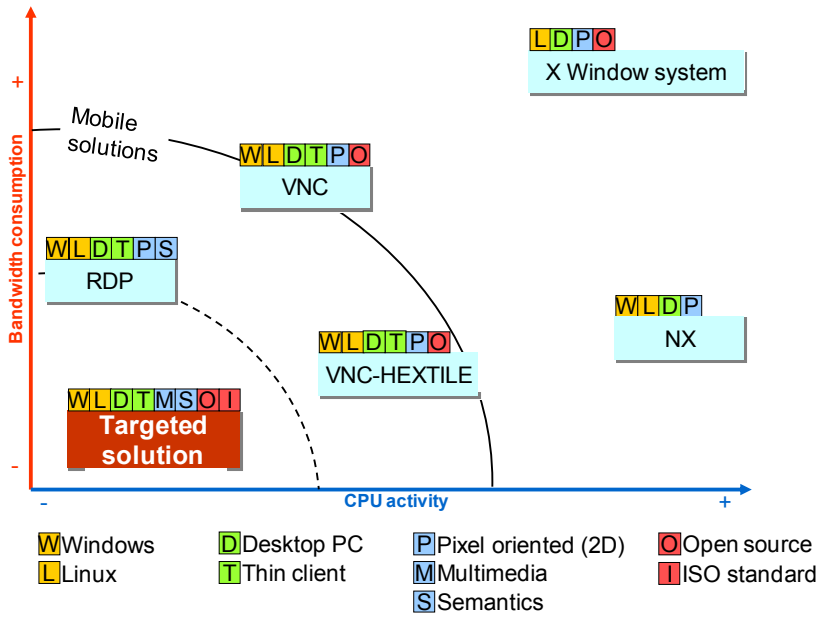


Figure 2.13. Comparison of the current remote display solutions

2.3 Collaboration technologies

In a real-time collaboration environment, an application is running on a server allowing one or more remote users to interact with its content, the structure of its representation or its behavior. The users connected to the collaborative application have the intention of consulting or influencing the shared content of the application; the process is referred to as to collaboration, see Figure 2.14.

Nowadays, the collaboration concept is much invoked, yet never addressed in its general form.

Figure 2.14. The concept of real-time user collaboration

The collaborative application exploits several methods for simultaneously updating all the connected users:

- *Asynchronously:* only one user can interact with the multimedia content at one time, thus consuming traffic only generated by one user.
- *Pooling:* multiple users can interact with the multimedia content at the same time, while the new updates are available to all the users after the application synchronize the content. In this way, the network traffic increases with the number of collaborators, but the interactions from the collaborators are not available in real-time.
- *Synchronously:* all the users interactions with the multimedia content are available to all of the collaborators in real-time, thus resulting in huge amount of Internet traffic while updating all the users in real-time with the collaboration actions.

The way in which these mechanisms are included in some illustrative real-life applications (document editing, gaming, social networking and instant messaging) is discussed in the sequel.

Documents editing

With the specification of the *Web 2.0* in 2005, new dynamic Internet applications have been introduced, enabling creation of the first collaborative applications. These applications are browser-based documents, written by HyperText Markup Language (HTML) tools. Although claimed to provide *simultaneous* text editing for all of the connected users, the various updates are actually reflected according to some *periodic data polling policies* (*e.g.* every half-minute). This technology promoted by *Google* (referred to as *Google Docs*), was released in February 2007, and is one of the few technologies offering real-time collaboration. In order for two or more users to collaborate over one existing document, the Google Docs generate an URL (an Internet address) where the collaborators can access/download the same collaborative application and edit the same document by using an HTML web browser.

Google Docs is restricted to basic and simple text editing functionalities (like typing, color selection, …) and does not allow the user to enjoy the fully functional desktop oriented text editing functionalities, like the ones featured by *Microsoft Office Word* for instance. Moreover the collaboration is application dependant, forcing the connected users to collaborate only on particular multimedia content and limiting them in defining their collaboration principles.

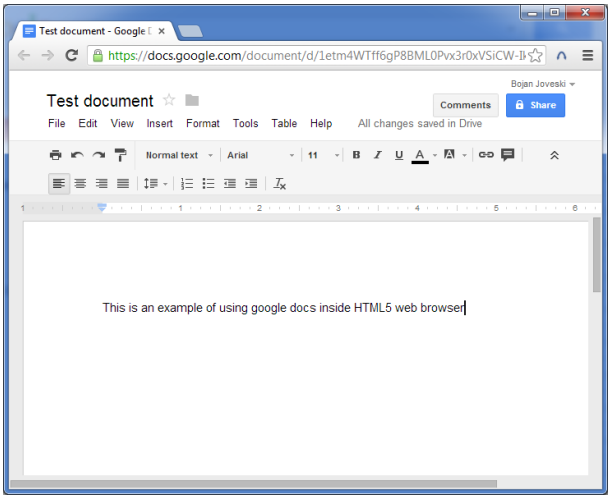


Figure 2.15. Creating a text document using the Google Docs

In 2009, Google introduced *Google Wave*, a collaboration environment, eventually to replace email and instant messaging; however, Google renounced this project in 2010, due to both insufficient user social networking adoption and to HTML inner limitations in terms of personalization of document styles (fonts, colors, headers,…), content structure (positioning of the figures/tables/images with respect to the text) and constant Internet connection during the editing period (for periodical document savings on the cloud).

Gaming

The modern, Internet connected games are always associated to the collaboration principles. They allow the users to play together the same game while not being physically on the same place. In this respect, the users have to install the same application (*i.e.* the software game itself) on their terminal (PC, smartphone, …) and to connect to the same collaboration server. Such an approach requires additional hardware resources at the client terminal, which might be an issue in the world of thin clients (limited to I/O resources).

While playing the game, the users actually interact with a local (terminal side) copy of the content while a synchronization process is updating both sides (the server and the connected clients) with the latest information about their actions (position, scene composition, …) [Blizzard, 2012], [Valve, 2012].

Figure 2.16. World of Warcraft (WoW) screenshot

Social networking

Social networking provides the communitarian user with the possibility of sharing heterogeneous multimedia content (video, music, journals, blogs, chatting, TV/radio stations,…). All these contents are usually available at any time and to any user terminal, but each user can access that content individually, without being aware of the other connected user's actions.

Consider the case of video streaming on YouTube, Figure 2.17. Generally, thousands of users access the same web page at the same time, in order to see the same video (stored on the server in a given video file). From the technical point of view, a different streaming session is asynchronously allocated to each user, thus allowing him/her to control only his/her own video stream. Consequently, two or more users cannot share the same visual experience at the same

time. If one of the users make a pause on the video, this will affects only his video stream, while the rest of the connected users will not be affected.

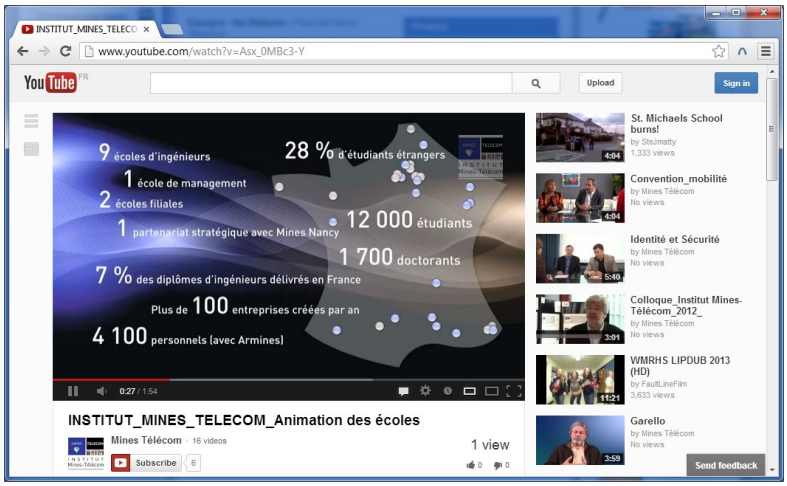


Figure 2.17. Video streaming on YouTube using www browser

The same mechanism governs practically all the multimedia content accessed on Internet via browser: each user can access that content (journals, blogs, chatting, TV/radio stations, …) based on a unique (none shared) session.

Instant messaging

The principles of exchanging messages trough internet between two or more users is also considered as collaboration. These types of collaborative applications allow two or more connected users to exchange multimedia content (text, images, videos and music) in real-time. The collaborative application captures the multimedia content generated by the user, transfers it to the collaboration server for further distributes it to all connected users. The received multimedia content is in “read only” state: it is not possible to change the received messages or to simultaneously create a new message in collaborative manner (with participation of the other users).

Note that instant messaging is not restricted to text messaging. Actually, video conferencing can be considered nowadays as the most popular type of “chat”, Figure 2.18, [Skype, 2012], [FaceTime, 2012].

Figure 2.18. Video conferencing using Skype (*cf.* credit to [Skype, 2012])

Conclusion

Currently, a myriad of collaborative applications (sharing calendars, project management systems, workflow systems, knowledge management tools, …) are available. All of these applications offer certain level of collaboration: they grant access to the content (or to its local replica) on a time-sharing basis (synchronous or asynchronous). Traditionally, their multimedia content is either statically created (during the application development/initialization) or dynamically (by individual users). Moreover, the collaboration itself is ensured at the application/operating system levels.

However, to the best of our knowledge, no application fully covers all the collaboration aspects (see Table 2.1):

- *real-time collaboration:* simultaneous users interaction on a given content;
- *multimedia collaboration:* independent with respect to the multimedia data type;
- *content level collaboration:* independent with respect to the application/operating system, by enabling collaboration over non-collaborative applications;
- *open-source/open-standard support:* independent with respect to the device, by using of ISO standard.

Table 2.1. Current collaboration status

	Collaboration		
	Time line	**Data**	**Level**
Documents editing	polling	single	application
Gaming	synchronous	multiplied	operating system
Social networking multimedia	asynchronous	single	application
Instant messaging	asynchronous	Single	Application

These limitations are considered in our study as the requirements for the new standard technologies to emerge.

2.4 Conclusion

The state-of-the-art investigations reported in the present Chapter (Section 2.1, 2.2 and 2.3) bring to light the following main limitations of the existing solutions for serving the design of a collaborative remote viewer for multimedia thin clients.

Table 2.2. Illustrations of the current limitations of the existing technologies

Constraints	Challenge	Current limitation
User Expectancies		
Multimedia	True multimedia content on the client side	Image (sometimes image&graphics)
Collaborative experience	Full collaboration support	No support at the content level, dedicated mechanism
Mobility		
Time-variant network bandwidth & latency	Real-time compression algorithm for multimedia content	*Downlink:* • compression algorithm based on regions of interest in images *Uplink:* • static compression for user messages
Market acceptance		
Terminal/OS proliferation	Terminal independency	Terminal/OS dependent solutions
Community support	Open source	Proprietary *vs.* open source

Chapter 3. Advanced architecture

Cette thèse propose une architecture basée sur la gestion sémantique du contenu multimédia pour définir des systèmes de rendu distant avec fonctionnalités collaboratives.

Le principe consiste à représenter le contenu graphique généré par le serveur comme un graphe de scène multimédia interactif, enrichi avec des nouvelles composantes pour permettre la collaboration directement au niveau du contenu. Afin d'optimiser la compression du graphe de scène sous contrainte des conditions réseau variable en temps, un cadre méthodologique pour la gestion sémantique du graphe de scène a été conçu (brevet en instance). La compression des messages collaboratifs générés par les utilisateurs est réalisée grâce à un algorithme sans perte basé sur la construction dynamique, en temps réel, des dictionnaires d'encodage (solution brevetée).

Cette nouvelle architecture a été progressivement évaluée par la communauté ISO et ses nouveaux éléments collaboratifs sont actuellement acceptés comme des extensions à la norme CEI JTC 1 SC 29 ISO WG 11 (MPEG 4 BIFS).

3.1 Functional description

As explained above, traditional remote display solutions are based on the conversion of the original content into sequences of images, Figure 3.1. These images are subsequently compressed, transmitted and rendered according to image/video principles and tools. Each remote viewer application comes with its own means for capturing the user interaction at the level of the OS drivers. The present section goes beyond the image limitations and advances a semantic collaborative mobile thin client architecture, centered on the MPEG-4 interactive multimedia scene technologies, Figure 3.2. In order to benefit practically from such technologies, a scene Scene-graph Management Module is designed and implemented. The content is then compressed and transmitted, according to open-standard/open-source tools. On the client side, the user events (key strokes, mouse clicks, *etc.*) are captured in an ISO standardized manner and are subsequently managed by an architectural block devoted to this purpose.

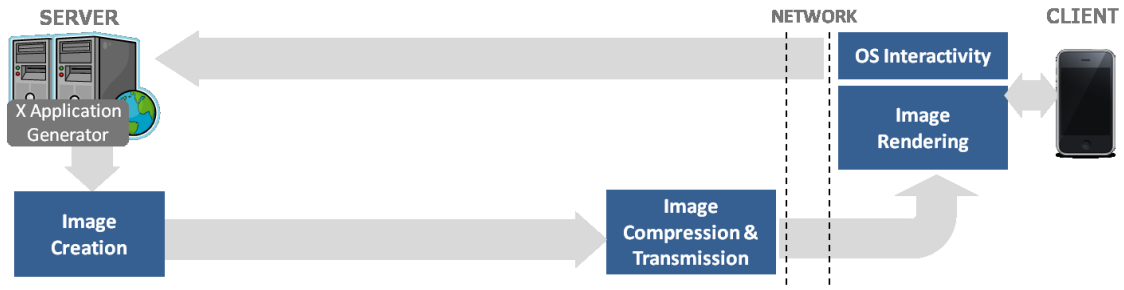


Figure 3.1. State-of-the-art architectural framework for mobile thin client remote display

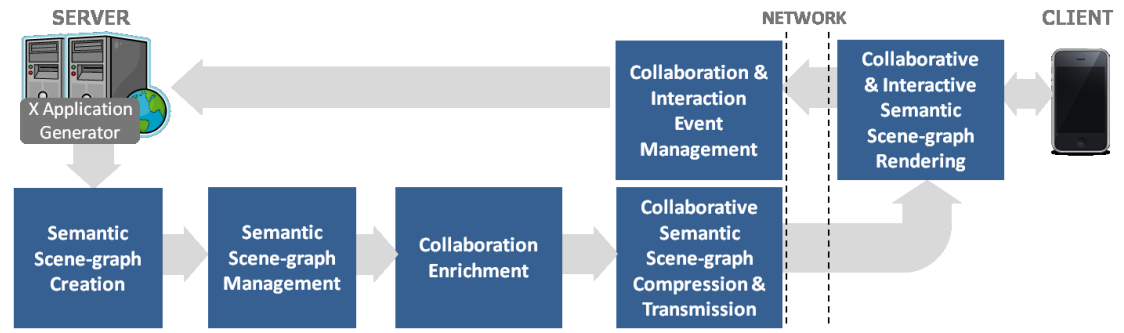


Figure 3.2. Advanced architectural framework for mobile thin client remote display

The application generator creates X graphical content that is to be presented at the client; it corresponds to the traditional application (be it text editing, www browsing, …) which is kept unchanged (*i.e.* from the application point of view, our architecture is completely transparent). Figure 3.2 explicitly considers X applications running on Linux servers; however, the architecture is general and can be instantiated in any OS, like Windows or Apple, for instance.

The Scene-graph Creation module performs three tasks. It detects the graphical primitives generated by the X application, parses them and subsequently translates them into a multimedia scene-graph preserving not only the multimedia content but also its semantic. The Scene-graph Creation module was designed to represent all the content generated by any X legacy application, by a semantic multimedia scene-graph, without changing the application. The underlying technical challenges are related to the completeness (*i.e.* the possibility of converting all the visually relevant X primitives) and flexibility (*i.e.* the possibility to integrate future X extensions with minimal impact in the architecture). To our best knowledge, no work on that direction was already reported.

The Semantic Scene-graph Management module ensures the dynamic, semantic and interactive behavior of the multimedia scene-graph. In this respect, the previously created scene-graph is enriched with logical information concerning its content type, its semantic and its related time of evolution as well as with user interactivity. The Scene-graph manager provides a heterogeneous content which can be subsequently optimally compressed (the optimality refers here not only to the trade-off of visual quality-bandwidth but also to the CPU activity). The innovation is related to the specification of an algorithm enabling the dynamical updating of the scene-graph according to the network/client/server conditions (be them real-time or evaluated on a short history).

The Collaboration Enrichment module enriches the semantic scene-graph with bidirectional collaborative functionalities. To our best knowledge, this is the first time when the collaboration functionalities can be ensured directly at the scene level. Moreover, from the functional point of view, direct client to client connection, independent with respect to the connection type, is for the first time advanced.

The Collaborative Semantic Scene-graph Compression & Transmission module is in charge of the creation of a binary encoded stream (the compressed scene-graph) which is subsequently streamed live to the client. The technical challenge is related to the flexibility of the transmission mode (unicast, broadcast, multicast) and of the transmission protocol.

The Collaboration and Interaction Event Management maps the user events back to the application, thus ensuring the server-side interactivity and contributing to the scene-graph management process. As in the Scene-graph creation module case, its technical challenges are related to the completeness of the solution and to its flexibility.

The Collaborative and Interactive Semantic Scene-graph Rendering is ensured by a multimedia player able to captures the user interactivity and lets the local interactive scene-graph handle it. The user interaction is captured in a standard way, at the scene-graph level while the rendering process demands in terms of CPU activity should not exceed the objective limits set by the nowadays thin clients.

The *Network* ensures the traffic from the server to the thin client (the streaming of the interactive scene-graph) and *vice-versa* (information concerning the user interaction).

3.2 Architectural design

This architectural framework is instantiated on a Linux virtual machine (VM) as a server and on a smart phone as a thin client, Figure 3.3. The actual implementation considers a server based on the Ubuntu installation accommodating the server components and a Windows mobile thin client accommodating an MPEG-4 player. The network is established by using a wireless protocol (the actual implementation considered a Wi-Fi 802.11g network). Note that the architecture presented in Figure 3.3 was incrementally advanced and evaluated in our publications [Mitrea, 2009], [Joveski, 2010], [Joveski, 2011], [Simoens, 2012], [Joveski, 2013].

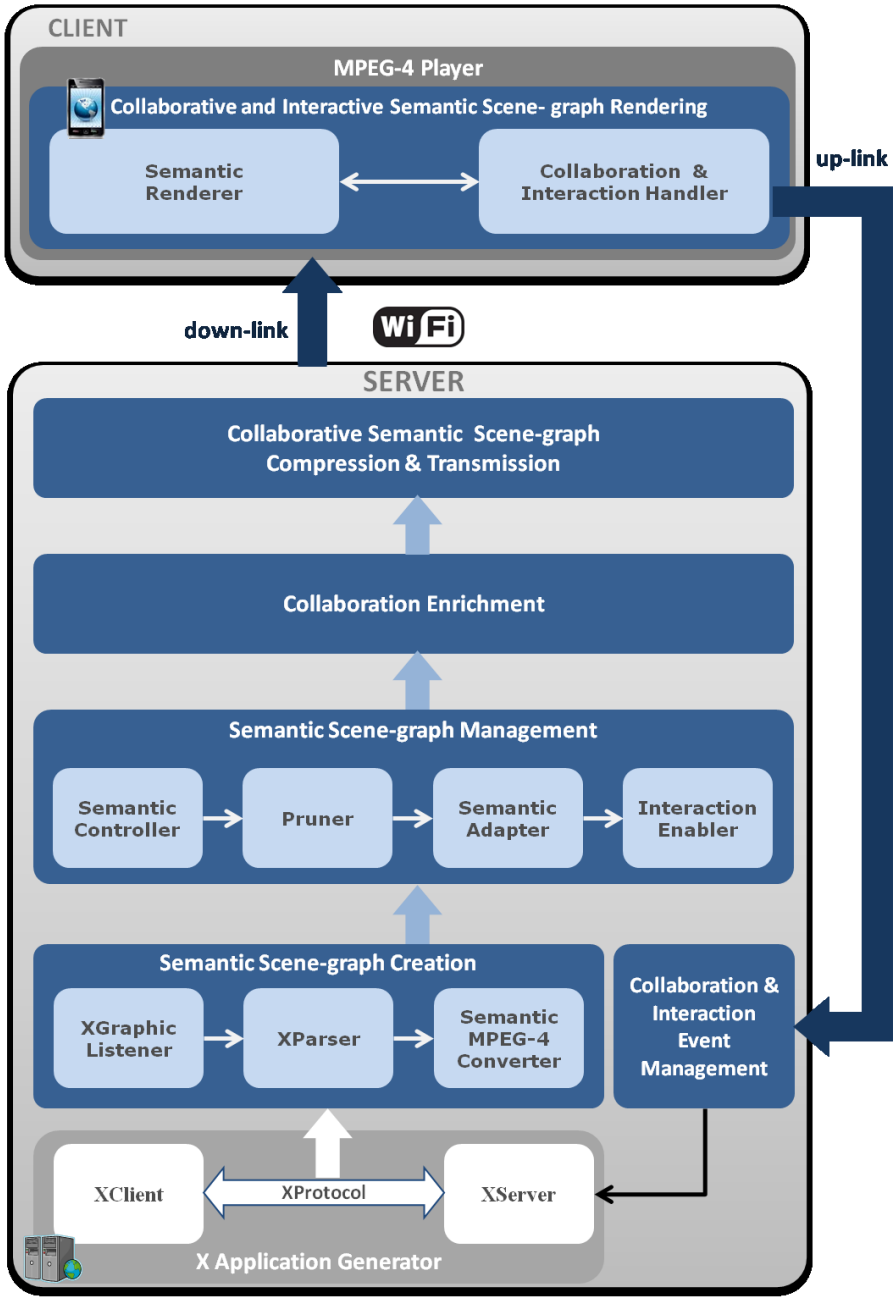


Figure 3.3. Detailed architectural framework

3.2.1. Server-side components

X Application Generator

This module is implemented by an X window system (XServer, XClient and XProtocol) and exploited by traditional applications (text editing, www browsing, …). In order to cope with the backward compatibility constraint, the X window system is kept unchanged during the present study.

Scene-graph Creation

The scene-graph Creation is implemented by three blocks, each performing one of the previously described tasks: the XGraphic Listener, the XParser and the Semantic BiFS/LASeR convertor.

XGraphic Listener

Located between the XClient and the XServer, by listening on a socket through which they communicate using the XProtocol, the graphic listener intercepts the X messages and passes the results to the XParser. By developing the XGraphical Listener as an independent architectural component encapsulated in a thread (light-weight process), it becomes completely transparent to both XClient and XServer. Moreover the transparency and the independence of this module allow several X applications to run simultaneously without any limitations (all the graphical output generated by the running applications are to be captured in real-time by the XGraphic Listener). The backward compatibility and the Unix-based OS independence are jointly ensured without application modification or development of a driver module. Moreover, no functional limitation is induced by listening to the XProtocol instead of intercepting the visual content directly at the XClient side: all graphical information is available at the protocol level and no network overcharge is produced (the XClient and the XServer run locally).

While the X Application is running it generates visual and semantic information related to the graphical content and consequently, the architecture presented in Figures 3.2 and 3.3 require no sophisticated segmentation/tracking/scheduling algorithms.

XParser

This component was developed for the parsing of the XProtocol in order to extract the graphical requests and their related semantics to be presented to the Semantic BiFS/LASeR converter.

In its current status, exploited by the opens-source communities, the XProtocol is composed of thousands of graphical requests/replies and their extensions. Usually the extensions, created by the communities to customize their visual environment and enhance their user experience, are intended to contain multiple requests. But, in its general structure, the basic set of only 119 graphical requests, (defined by the core of XProtocol) is sufficient for visualization of any application. Besides the diversity in the requests, the XParser executes the following four steps:

(1) detect the request ID; (2) get the length of the request; (3) read the values; and (4) pass this information to the *Semantic BiFS/LASeR Converter.*

Just as an illustration, consider the following case in which we are interested in the PolyRectangle request; its complete X Protocol syntax is:

```
1 bytes  67 opcode      // the request message ID
1 bytes  unused
2 bytes  3+2n requestlength    // the length of the request message
4 bytes  DRAWABLE drawable     // the parent of the graphic primitive
4 bytes  GCONTEXT gc      // the description of the rectangle material
8n bytes LISTofRECTANGLE rectangles   // list of rectangles with position and size
```

Code 3.1. X Protocol description of polyRectangle syntax

In order to parse this message from the X protocol, the following code can be used:

```
drawable = x11application->getUInt32(&(message[0]));
graphicalContent = x11application->getUInt32(&(message[4]));
noRectangles = x11application->getUInt16(&(header[2] ))- 3 / 2;
for (i=0; i < noRectangles; i++)
{
        x=x11application->getUInt16(&message[8 + 8 * i ]);
        y=x11application->getUInt16(&message[8 + 8 * i + 2]);
        width=x11application->getUInt16(&message[8 + 8 * i + 4 ]);
        height=x11application->getUInt16(&message[8 + 8 * i + 6]);
}
```

Code 3.2. Syntax of parsing polyRectangle by the XParser

Semantic MPEG-4 Converter

Each X request intercepted by the parser is mapped to a function which converts it to its BiFS/LASeR counterpart: all of the 119 core protocol X visual requests/replies (rectangle, line, circle, *etc…*), text and images have already been successfully converted. Assuming the X window system will be extended in the future with other graphical primitives, this component should also evolve so as to cope with these updates. Although it is not possible today to foresee the syntax of these extensions, the possibility of converting them in BiFS/LASeR elements is guaranteed even when no straightforward MPEG counterparts would be available: as a worst case scenario, these future graphical elements would be rendered and the corresponding pixel maps would be included in the MPEG scene-graph.

Note that the Semantic BiFS/LASeR Convertor also allows the semantic information about the X content to be converted for use in the management of the MPEG-4 scenes.

When considering the example above, the following BiFS conversion expressed in XML format is represented in Code 3.3.

```
Transform2D {
     translate x y
     children [
          Shape {
               appearance Appearance {
                    material Material2D {
                    }
               }
               geometry Rectangle {
                    size width height
               }
          }
     ]
}
```

Code 3.3. XML description of BiFS conversion of a rectangle

This corresponds to the following LASeR description (SVG format), see Code 3.4:

```
<rect width="" height="" x="” y="” style="fill:rgb(,,);stroke-width:1; stroke:rgb(,,)"/>
```

Code 3.4. SVG description of LASeR conversion of a rectangle

Note that in contrast to the BiFS situation, not all the X graphic primitives have a straightforward conversion in LASeR. For instance, LASeR makes no provision for describing raw images, which are part of the *XProtocol* generated by the *PutImage* request. In such a case, more elaborated scene management mechanisms are provided. For instance, in order to convert the *PutImage* primitive, the related pixel buffer corresponding to a raw image is first converted into a png/jpeg binary buffer. This buffer is base64 encapsulated and mapped to the LASeR *Image* node.

The complete conversion dictionary used for converting the XProtocol requests into their MPEG-4 BiFS and LASeR counterpart is presented in Appendix III.

Of course, in our study, BiFS and LASeR are not operating at the same time (they are alternatively enabled, in order to ensure a comparison of their performances).

Semantic Scene-graph Manager

As previous mentioned this component ensures the dynamic, semantic and collaborative behaviors of the MPEG-4 scene-graph. The dynamic and semantic evolution of the scene-graph can be managed by combining the information generated by application with the semantic tagging of the scene-graph elements and a prescribed set of logic rules concerning the possible re-usage of the most common graphic elements (*e.g.* menus, icons, …) and/or the adaptation of the content to the actual network client conditions. The current module implementation is based on four main blocks: *Semantic Controller*, *Pruner*, *Semantic Adapter* and *Interaction Enabler*.

Semantic Controller

By exploiting the semantic information about the elements composing the scene-graph, some *a priori* hints about their usage can be obtained. For instance, when typing, the most frequent letters/words represent the most frequent scene updates. Hence, an important network bandwidth gain would be achieved when caching this content on the client side for its re-usage. This gain would be even more important when considering menus, icons or particular images during www browsing.

Regardless its type (text editing, www browsing, …) each X Application can periodically generate identical visual content. Such a case does not only occur when refreshing the screen but also when dealing with some fixed items (frequent letters/words typing, icons or menus displaying, *etc*.) or with repeated user actions (mouse over, file open, document save, *etc*.). Consequently, significant bandwidth reduction is *a priori* likely to be obtained by reusing that content directly on the client side, instead of resending it through the network. However, in order to take practical advantage of this concept, a tool for automatically detecting the repetition of the visual content and for its particular management should be designed. Figure 3.4 presents an example corresponding to the particular case of image re-using. Similar mechanism can be deployed for each type of multimedia content.

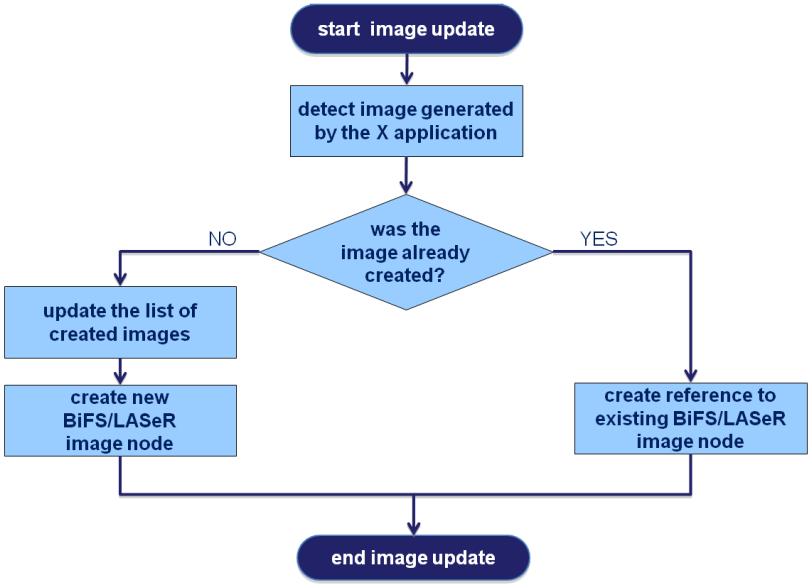


Figure 3.4. Flowchart for image management

The scene updating starts by detecting an image generated by the application output as a result of user interaction.

Then, the existence of this image in the scene-graph is checked. While conceptually this task is simple (a comparison between two images), in practice, when handling hundreds of images of

large sizes, a more elaborated decision making mechanism should be provided. We considered a two-level decision rule defined in order to assure fast and correct image scene update:

1) *compute the MD5 hash*, for each and every new image in the scene-graph; this way, the comparison between two images of any size can be done on a fixed-length data string of 128 bit.
2) *compare the image basic info:* by comparing the image width, height and position within the existing image database; if a match is found, compare the MD5 hashes of the images positioned at the same location; if not, search all the MD5 database.

Finally, if image already exists in the scene, a simple reference (pointer) to the corresponding image is created. Otherwise, the hash database is updated and the new image is placed in a new node (or, in several nodes) in the scene-graph.

Pruner

As the thin client has limited memory resources, a pruning mechanism, controlling the data persistency is required.

The thin client memory is limited by its hardware resources. While the content reusing would suppose, as a limit case, the caching of all the visual content previously generated in a session, the limited memory resources requires a mechanism for dynamically adjusting the cached information. Several implementation choices are available, from a fixed time window to more elaborated decisions based on actual frequency of usage or on the content semantic.

Hence, for thin clients, the image reusing should be restricted in time, thus removing all unused content at the thin client after given time. In our implementation, we combined some temporal and spatial information about the cached images: assuming some images in the scene are not visible (*i.e.* they are covered by other visual elements) for more than τ seconds (in the experiments, $\tau = 180$ seconds), they are removed from the scene and the image database is updated accordingly. Of course, different decision making rules can be implemented here: while directly impacting the system performances, they would not affect the architecture generality.

The functional workflow of this block is presented in Figure 3.5.

Generally removing a node from the scene is an easy task but removing the correct scene node requires particular actions. The scene updating starts by detecting the image scene update, an output from the S*emantic Controller*.

The first thing after detecting the image scene update is to check whether this update covers an image already in the scene. If it is the case, then check whether the image already existing, is not being used in the scene, and it is present at the scene for more than τ seconds. After this second check, if the condition is fulfilled the database of created images is updated and a delete scene BiFS/LASeR command is sent as a scene update.

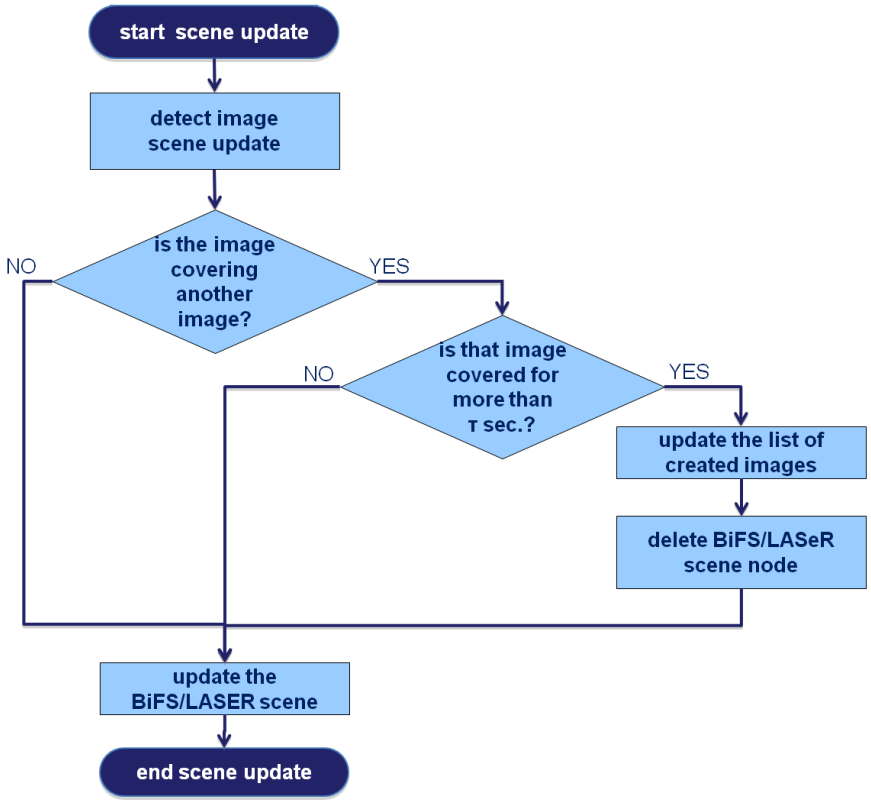


Figure 3.5. The Pruning mechanism, exemplified for image content

Finally, the BiFS/LASeR scene is updated so as to take into account these changes: adding a new image / pointer to an image and remove some old images.

Semantic Adapter

While in the traditional approach of the scene-graph creation, for each user terminal and its network conditions, a new scene-graph is created, Figure 3.6, we designed a new concept and method for dynamic real-time scene-graph adaptation, Figure 3.7.

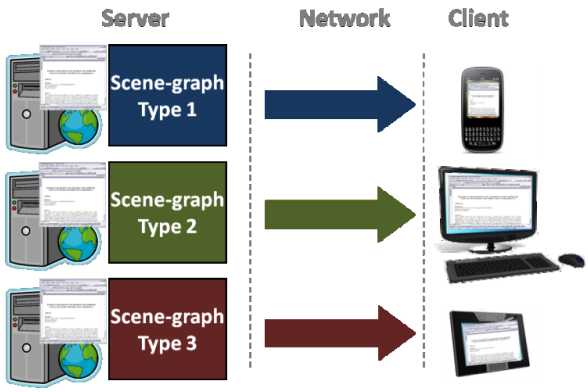


Figure 3.6. Traditional scene-graph creation

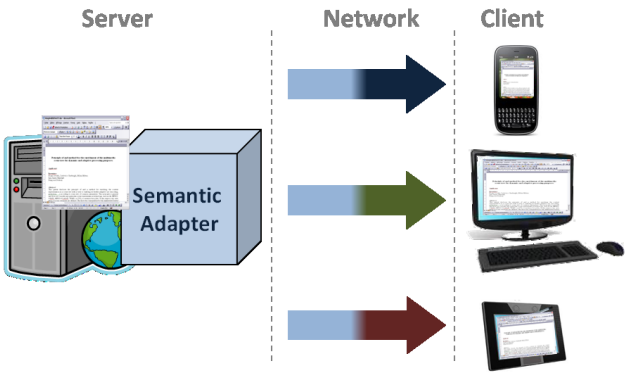


Figure 3.7. Advanced scene-graph adaptively created

Today, the scene tree representation technologies (*e.g.* MPEG-4 BiFS) make no provision for the scene-graphs to be dynamically and adaptively processed with a view to its transmission, when the user changes its terminal. There is also no provision for allowing the same scene to be encoded with time dependent parameters, according to the time/environment-dependent bandwidth constraints intrinsically connected to a mobile device.

The existing solution for these problems is static and consumes a lot of memory. It is based on a lists of all the possible combinations (device configuration, network condition, *etc.*): for each possible combination, one different scene-graph is created. Hence, the server is overcharged by storing all the combinations of the scene-graphs, ready to be streamed to the client when needed. Moreover, the number of the combinations are practically unlimited, whereby the server cannot make a prevision in advance.

In order to represent this traditional approach, we grouped the scene-graph processing in three levels, see Table 3.1: initialization, computing and rendering.

Initialization is the process of preparation of the scene-graph description, including the appliance of some additional constraints and parameters. During this process, according to each constraints and parameter combination, a new scene-graph is created. At the end of this process there is a list of scene-graphs ready to be computed is provided.

According to the user constraints, the *Computing* process makes the decision of the appropriate scene-graph ready to be streamed to the client.

The user terminal receives the scene-graph, and subsequently displays it during the *Rendering* process.

Table 3.1. Traditional MPEG-4 scene-graph processing

	The flowchart	The mathematical expression	The practical relevance
Initialization	$GS = (S, E, \Pi)$	The Graph (*GS*) is defined as a set of nodes (*S*), on which some topological relations are defined (represented by a set *E*) depending on some parameters (Π).	The multimedia scene is represented as a scene-graph.
	$\xi : t \mapsto = \xi(t) = \xi_t$	The constrains in the environment are defined as functions of time $\xi(t)$.	The space of all possible constraints to be applied to the scene-graph is considered.
	$\psi : \xi(t) \mapsto \psi[\xi(t)] = \Pi_t$	All the values possible for the $\psi[\xi(t)]$ functions, are considered.	In order to apply the constraints to the graph, some strategy functions are used to set the scene parameters, at the server side.
	$\left[\begin{matrix}(S_1, E_1)\\(S_2, E_2)\\.\\.\\(S_n, E_n)\end{matrix}\right\}, \; n = card(\xi_t)$	A different scene-graph for any possible configuration in the constraint space is created.	A different multimedia scene is available for any possible configuration in the constraint space
Computing		The actual value of the $\xi(t_i)$ is measured and the corresponding (S_i, E_i) is selected.	The scene-graph corresponding to the constraints
Rendering			The scene-graph (S_i, E_i) matched to the client conditions can be rendered.

Our study, follows a different approach and advances the solution presented in Table 3.2. (Patent pending, [Mitrea, 2012], see Appendix I)

Initialization is the process of preparation of the scene-graph description, including the appliance of some additional constraints and parameters. During this process, a detection of a sub-graph is

performed according to each constraint and parameter combination. At the end of this process a list of parameters ready to be computed for a new scene-graph creation is available.

Table 3.2. Advanced scene-graph adaptation

	The flowchart	The mathematical expression	The practical relevance
Initialization	$GS = (S, E, \Pi)$	The Graph (*GS*) is defined as a set of nodes (*S*), on which some topological relations are defined (represented by a set *E*) depending on some parameters (Π).	The multimedia scene is represented as a scene-graph.
	$\varphi(GS) = \left(\varphi(S), E\big\|_{\varphi(S)}, \Pi\big\|_{(E\|\varphi(S)),\varphi(S)}\right)$	The sub-graph $\varphi(GS)$ is defined as a function $\varphi : S \mapsto S$, identifying a sub-set of *S*, the restriction of *E* to this sub-set and a set of parameters $\Pi\big\|_{(E\|\varphi(S)),\varphi(S)}$.	The sub-graph to be adaptively processed is selected .
	$\xi : t \mapsto = \xi(t) = \xi_t$	The constrains in the environment are defined as functions of time $\xi(t)$.	The space of all possible constraints to be applied to the scene-graph is considered.
Computing	$\psi : \xi(t) \mapsto \psi[\xi(t)] = \Pi_t\big\|_{(E\|\varphi(S),\varphi(S))}$ $\left[S, E, \left[\Pi_t\big\|_{(E\|\varphi(S),\varphi(S))} \cup \Pi\big\|_{(S-\varphi(S)),(E-E\|\varphi(S))}\right]\right]$	Compute the $\psi[\xi(t)]$ functions and the corresponding scene-graph matched to the actual client network conditions.	In order to apply the constraints to the graph, some strategy functions are used to change the scene parameters; this step can take place on the server side, somewhere in cloud or even on the client, according to the targeted use case.
Rendering			The client renders its adapted scene-graph previously computed

According to the user constraints, the *Computing* process refers to generating the scene-graph matching the parameters defined in the I*nitialization* process (according to the network condition, user terminal hardware limitations).

After the user terminal receives the matched scene-graph, the *Rendering* process displays the scene-graph in a fast and customized way.

With designing and implementing of this block, we can identify the following functional enablers for a semantic multimedia remote viewer for collaborative thin clients:

- adaptive encoding of a unique scene tree for different clients, depending on the client's hardware/software capabilities and/or on the user profile;
- single decompression is sufficient when reusing compressed data (*e.g.* images) on the client side;
- simplification of the scene management by avoiding node duplication when multiple encodings are needed;
- single point of control for complex scene update;
- dynamic update to the processing parameters;
- improved flexibility without adding complexity on the client side.

Interaction enabler

In order to ensure the user interactivity mechanisms, basic MPEG-4 elements, referred to as *sensors* [BiFS, 2006], are considered in the multimedia scene-graph. These *sensors* with combination of JavaScript functions are the main support for capturing the user events at the scene-graph level, and further processed (locally or remotely). As an example part of the C language code used by this block for detecting the mouse click and mouse position is presented in Code 3.5.

```
static const char* buildJavascript(X11toBIFSLib* lib) {
       const char* javascriptTemplate = "javascript:"
              "function initialize() {"
              "  clickRequested = false;"
              "  pressedValue = false;"
              "  havePosition = false;"
              "}"
              "Function MousePosition(value) {"
              "  MouseX = Math.round(value.x);"
              "  MouseY = Math.round(- value.y);"
              "  havePosition = true;"
              "  if (clickRequested) {"
              "    MousePressed(pressedValue);"
              "  }"
              "}"
              "Function MousePressed(value) {"
              "  if (1) {"
              "    SCCommand = 'M' + MouseX + ','+ MouseY+','+(value ? 1 : 0);"
                 SCTrigger = 1;"
```

```
                "     havePosition = false;"
                "   } else {"
                "     clickRequested = true;"
                "     pressedValue = value;"
                "   }"
                "}"
        "}";
}
```

Code 3.5. Part of the C language code enriching the scene-graph with JavaScript functionality for mouse click

Collaboration Enrichment

Within the scope of the above-described global architecture, it was necessary to extend the capabilities of the MPEG scene description technologies to allow control of the collaboration subsystem directly from the scene tree graph. In order to ensure the user interactive collaborative mechanisms, the new collaborative extension with the use of the standards MPEG-4 elements referred to as *sensors*, are considered in the multimedia scene-graph.

A basic architecture based on the ISO MPEG-4 BiFS standard, describing a complete collaboration system, providing for the generic needs of multi-user and collaborative applications, is represented in Figure 3.8.

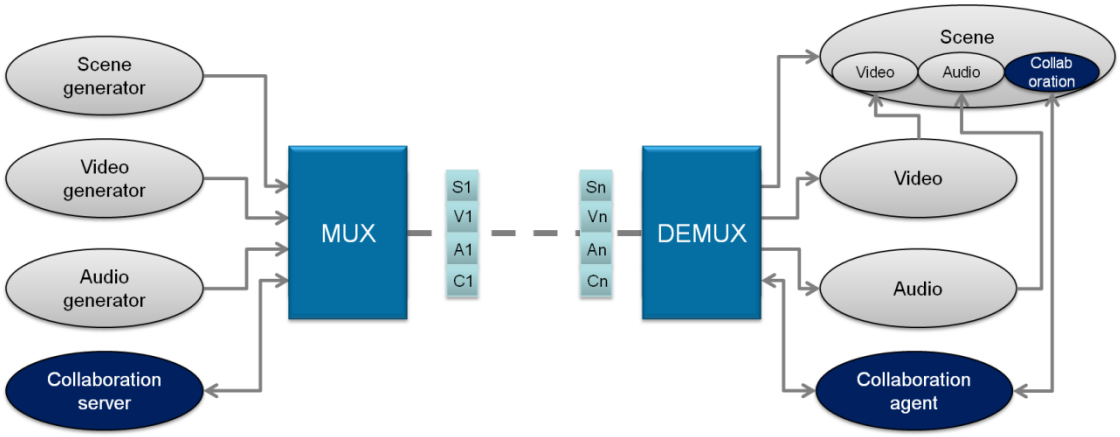


Figure 3.8. General architecture for collaborative scenes

The general functionality of the current day application based on MPEG scene description technology considers only the multiplexing and de-multiplexing of scene, video, audio elements without any provision of collaboration. It can be exemplified as following:

1. on the content generator side, we have the various elements (scene/audio/video/…) composing the scene-graph;
2. these elements are multiplexed and transmitted to the player as MPEG-4 streams;
3. they are then de-multiplexed and some components may require to be decoded;
4. the scene is reconstructed and rendered and consequently becomes operational.

To achieve the collaborative dimension of the overall application this elementary architecture requires extension to include:

1. Collaboration server, on the server side, responsible for the propagation of collaborative messages and state between collaborating endpoints;
2. Collaboration agent, on the player side, counterpart to the collaborative server.

For enabling the collaboration functionality the following node description is specified:

```
CollaborationNode {
    eventIn       SFBool      triggerIn
    eventOut      SFBool      triggerOut
    exposedField  SFBool      Enable          FALSE
    exposedField  MFString    url             []
    exposedField  SFString    Message         ""
    exposedField  SFString    connectionType  ""
    exposedField  SFBool      Bidirectional   TRUE
}
```

Code 3.6. Technical description of the CollaborationNode

The *CollaborationNode* allows a scene to initiate the exchange collaborative messages with a Collaborative Server through the Collaboration Agent. Messages are exchanged under the control of and in response to collaborative events (both asynchronous and synchronous), be they generated from within the scene or from the server. On the one hand, at the scene level, events can be generated by the actual scene description, user interaction, scripts, collaborative server messages … On the other hand, the collaboration server generates events according to messages received from other collaborative agents of the same scenes or according to collaborative application logic.

The *CollaborationNode* is processed either when *triggerIn* or *triggerOut* receives a TRUE event and *enable* is TRUE. When the *CollaborationNode* is processed, the *messages* are sent to the server(s) indicated by the specified *url*. The *message* field contains the message that is transmitted to the *url* defined. The syntax and semantics of the *message* string are application specific and not specified.

The *connectionType* field provides information about the channel established between the collaborative server and scene (like UDP or TCP, for instance). The *bidirectional* field is TRUE for bidirectional communication and FALSE otherwise.

At the output from this block, interactive collaborative semantic multimedia scene-graph is ready to be streamed.

This enriched scene-graph with collaboration can also ensure direct collaboration between two or more clients. This will be further explained in the *Collaborative Interactive and Semantic Scene-graph Rendering* module located at the client-side.

Note: the *CollaborationNode* enables bidirectional users collaboration; this technology, developed in our study, is accepted by the MPEG-4 community and is expected an final ISO standard in 2013.

Collaborative Semantic Scene-graph Compression & Transmission

This module integrates the GPAC libraries for the binary encoding of the BiFS/LASeR graphical content [GPAC, 2012], [Concolato, 2008] and the streaming support from the LIVE555 Streaming Media [Live555, 2012]. The input to the streamer is BiFS/LASeR MPEG-4 stream content while its output is sent to the thin client by using RTSP/RTP. Note that nowadays the GPAC is the only open-source, publically available reference software framework for BiFS/LASeR; hence, its usage is implicitly compulsory. However, the use of LIVE555 and of RTSP/RTP was an implementation choice guided by their versatility (connection mode, usage of the protocol and streaming buffer control). According to the targeted application, the streaming tool can be changed, without affecting the rest of the architecture.

Collaboration and Interaction Event Management

It receives the user events, sent through the up-link (see Section 3.6.3 below), by the *Collaborative* and *Interactive Semantic Scene-graph Rendering* module at the client side. In the current implementation, the user interactions like keyboard and mouse/touch screen events are mapped to the users ID, in order to track and process the users interaction accordingly. The *Collaboration and Interaction Event Manager* converts these events into the syntax required by the XServer which ensures the server side interactivity mechanisms, *i.e.* it updates the X Application (XServer updates). Moreover, this module is enabled for handling the multi user collaboration, meaning handling all the user interaction received from the collaborators included in the process of collaboration.

For instance, the server side code for handling a mouse-click event by converting it into the X syntax is represented by the following Code 3.7.

```
if(leftButton==0) {
  //getting the time of the day
  gettimeofday(&currentTime,NULL);
  //setting the last click moment
  lastClickTime = currentTime;
  //Posting the button event
  conv->PostButtonEvent(MT_BTN_LEFT,MTBUTTON_DOWN,&currentTime);
  conv->PostButtonEvent(MT_BTN_LEFT,MTBUTTON_UP,&currentTime);
}
```

Code 3.7. C language code for posting the mouse left button click on the application

3.2.2. Client-side components

Collaborative and Interactive Semantic Scene-graph Rendering

Hosted by the GPAC MPEG player (part of GPAC multimedia solution package), its functionalities are mapped to two blocks, namely *Semantic Renderer* and *Collaboration and Interaction Handler*.

Semantic Renderer

The scene-graph received through the down-link is decoded, the multimedia scene-graph objects and their semantics are recovered and classified into visual and non-visual content. The visual content is displayed by using the basic GPAC libraries. The non-visual content (collaborative interaction sensors and JavaScript) are subsequently forwarded to the *Collaborative Interaction Handler*. In order to enable rendering of all the semantic multimedia content and the collaborative user events the GPAC libraries had to be modified in our study.

Collaboration and Interaction Handler

This component has three main functionalities. First, by using the MPEG-4 interaction mechanisms, it captures the user events. Secondly, it makes a decision about processing the events locally (at the client-side) or remotely (at the server-side). Finally, it handles the collaboration.

After rendering the enriched scene-graph sent from the server, this block is able to capture all the user events. The events captured can be not only simple keyboard press and mouse click, but also complex multi touch events. After the event is captured, this block decides whether it is client-side or server-side event. In the former case, it executes the corresponding scene-graph update, allowing the *Semantic Renderer* block to display this scene update without contacting the *Collaboration and Interaction Event Manager* at the server side. In the latter case, it forwards the event to the *Collaboration and Interaction Event Manager* by one of the two mechanisms explained in Section 2.1.3. This module also required the modification of the GPAC reference software, so as to support the *ServerCommand* specified by the MPEG-4 standard but, to our best knowledge, never implemented yet (at least in an open-source, publicly available software).

According to the scene-graph composition, the collaboration functionality is enabled, allowing the user actions to be collaboratively processed. During the process of collaboration, each user generates lots of data (presence, users' actions, scene-graph updates, …); all of them, are multiplied by the number of users connected. In order to optimize the data transmission through the network, we patented a lossless encoding algorithm for this type of data.

The method of encoding the data (patented solution [Marshall, 2012], see Appendix II) comprises the exchange between the transmitter and the receiver.

In order to demonstrate our algorithm, we will consider the XMPP (Extensible Message and Presence Protocol) encapsulation and transport of multimedia flows used and accommodated to MPEG.

The collaboration messages can be of three types:

- a message of a first type is having the presence status of the transmitting device;
- a message of a second type is having the information to a user of the receiving device;
- a message from a third type is having the metadata administration of a collaborative system in which the session is implemented.

For each of these three message types in this example, are used the following attributes:

- "To" which defines the message recipient;
- "From", which defines the message sender;
- "Type" that defines the semantics and is encoded with a predetermined value for each type;
- "ID" which defines information allows easy identification of the message by the collaborative application.

In the particular cases of the message of collaboration should also include:

- A field representative of the type of the message;
- One bit representing the use or not of an attribute in said message;
- Where appropriate, depending on the value of the binary element, a body with the index associated with the attribute shared dictionary.

By using this technology we are able to establish direct connection between two collaborators, as represented in Figure 3.9. Firstly, the collaboration server sends the enriched scene-graph with enabled collaboration through the initialization link. Secondly after the users process the scene-graphs they are able to establish direct link for collaboration. Finally, the initialization link can be closed or used for further collaborative messages, according to the administration constraints, user rights, *etc*. This makes the collaboration server to reduce the computational processing as not being active for all the collaboration time.

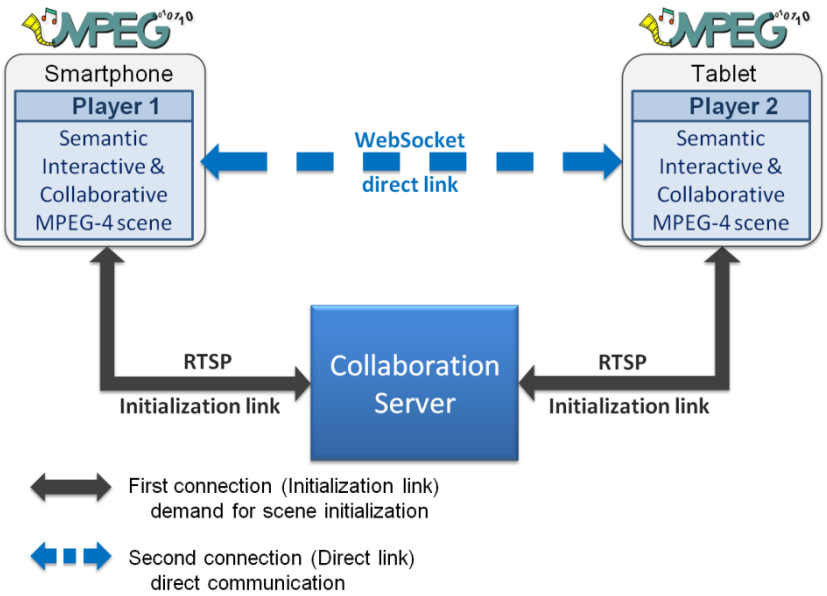


Figure 3.9. Direct client to client collaboration by using the collaboration node

3.2.3. Network components

The network components ensure the bidirectional transmission of the data between the client and the server. On the one hand, the live multimedia data are sent from the server to the client, through the downlink. On the other hand, the user interaction with the content is sent from the client to the server through the uplink.

The collaboration messages are by their very nature bi-directional, so they are exchanged through both the downlink and the uplink.

Down-link

The live multimedia data are sent through a channel managed by the RTSP/RTP over TCP (Real Time Streaming Protocol/Real Time Protocol over Transmission Control Protocol). In our study, the use of the TCP was an implementation choice rather than a technical requirement; should the applicative environment impose constraints on the use of this protocol, alternative solutions can be considered, as the popular TCP or as the emerging MMT (MPEG Media Transport) and DASH (Dynamic Adaptive Streaming over HTTP) MPEG standards [MMT, 2010], [DASH, 2011].

The collaboration messages are sent from the server to the client (or from the client to the client) through a WebSocket channel initiated by the newly standardized MPEG-4 BiFS *CollaborationNode*. This is an implementation choice and several other types of connections can be used with quite similar performances (*e.g.* XMPP).

Up-link

This channel is mainly used by the client in order to enable server-side user interactivity, according to the MPEG-4 mechanisms, by exploiting both *AJAX HttpRequests* and the *ServerCommand*. The former case is supported by the HTTP in conjunction with TCP. To the best of our knowledge, no study on the practical usage of the BiFS *ServerCommand* is reported in literature [GPAC, 2012]; hence, we considered both the TCP and UDP when dealing with the latter case. Note that as for the downlink, the protocol choice can be made according to the particular configuration in which the application is expected to work, without restricting the architectural generality.

The collaboration messages are generated by the mechanisms described in the downlink section.

3.3 Conclusion

Chapter 3 presents a comprehensive architectural framework able to take all the challenges related to the design of a semantic remote viewer for collaborative thin clients, see Table 1 (Abstract):

- *the true multimedia experience on the client side*, ensured by an architecture centered on the concept of multimedia scene-graph, affording an hierarchical representation of any type of content (text, audio, image, video, 3D, graphics);
- *full collaboration support*, obtained by specifying and standardizing ISO IEC scene elements enriching the multimedia scene with simultaneous collaboration, at the content level;
- *real-time compression algorithm for multimedia content on both downlink and uplink*, provided by real-time scene-graph adaptation mechanisms (patent pending), by the semantic management of the scene-graph and by the dynamic encoding of the collaboration messages (patented solution);
- *terminal independency*, guaranteed by the ISO compliance of the advanced architecture;
- *community support*, made possible by the open source approach in the implementation of the architectural modules.

Chapter 4. Architectural benchmark

Le démonstrateur logiciel sous-jacent, dénommé MASC (Multimédia Adaptive Sémantique Collaboration) est implanté par une approche logiciel libre. MASC a été comparé à des solutions fournies par des industriels comme VNC (RFB) ou Microsoft (RDP).

Il a été démontré que: (1) MASC offre une haute qualité visuelle (PSNR compris entre 30 et 42 dB et SSIM supérieur à 0,9999), (2) la consumation de la bande passante *downlink* présente un gain de 2 à 60, tandis que la consumation de la bande passante *uplink* comporte un gain de 3 à 10, (3) la latence dans la transmission des événements générés par l'utilisateur est réduite par un facteur de 4 à 6; (4) la consumation des ressources de calcul côté client, bien que plus grande que dans le cas RDP, est réduite par un facteur de 1,5 par rapport à la VNC RFB.

4.1 Overview

While a large number of studies reported in the literature [Beg, 2002], [Calluccio, 2005], [Basso, 2002] already evaluated the MPEG technologies performances when serving all types of video content applications, the present study is oriented towards two real-life, native X window applications, namely the gEdit [gEdit, 2012] text editor and the Epiphany [Epiphany, 2012] www browser. The former illustrates applications generating simple graphics, icons and text (development, office, e-mail, chat, *etc.*). The latter is an incremental stage, at which (high quality) images and more complex graphics are also generated; hence, the content generated by Epiphany is representative not only for the www browsing but also for image editing, virtual map accessing or professional medical applications, for instance.

The underlying software demonstrator based on the architecture presented in Figure 3.3, referred to as MASC (Multimedia Adaptive Semantic Collaboration), is open-source implemented. This implementation supports the two MPEG-4 scene description languages BiFS and LASeR. In this respect, the architecture was implemented and benchmarked into three cases: (1) BiFS, (2) MASC-BiFS and (3) MASC-LASeR. The first case (BiFS) denotes the basic solution, in which the Semantic Scene-graph Management module is not enabled. The last two cases (MASC-BiFS and MASC-LASeR) denote the complete implementation of the architecture in Figure 3.3, where the Semantic Controller and Pruner are based on png compressed images (compression level 9 and image-depth 24). A discussion about the choice of png compression format is presented in Section 4.3

Note that MASC-BiFS and MASC-LASeR required the basic GPAC player to be adapted accordingly. In the sequel, these three MPEG-4 based solutions were benchmarked against three on-the-market mobile thin client technologies: basic VNC, VNC-HEXTILE, and the Linux implementation of RDP [XRdp, 2009].

The following Section 4.2, Experimental setup, details the experiments carried out by using the MASC software demonstrator, while the Section 4.3, Discussions, elaborates the importance of the blocks in the architecture and their influence on the overall performance. Section 4.4 discusses the MASC industrialization potential.

4.2 Experimental setup and results

The experiments were successively conducted so as to assess the four main properties of the MPEG-4 mobile thin client remote display: the visual quality of the rendered content, the downlink bandwidth consumption, the user interactivity efficiency, and the CPU activity at the thin client side.

These experiments were carried out on the following setup:

- *server*: a desktop platform, with Intel Xeon CPU, 3.2 GHz, 4GB of RAM, 5400rpm 500GB of HDD;
- *client*: an HTC HD2 smartphone, with Snapdragon™ CPU, 1GHz, 448MB of RAM, 768 MB internal memory;
- *network*: an USB Wi-Fi 802.11g access point directly connected to the server; the mobile client located at distance varying between 2 meters and 5 meters from the access point, with a direct line of sight.

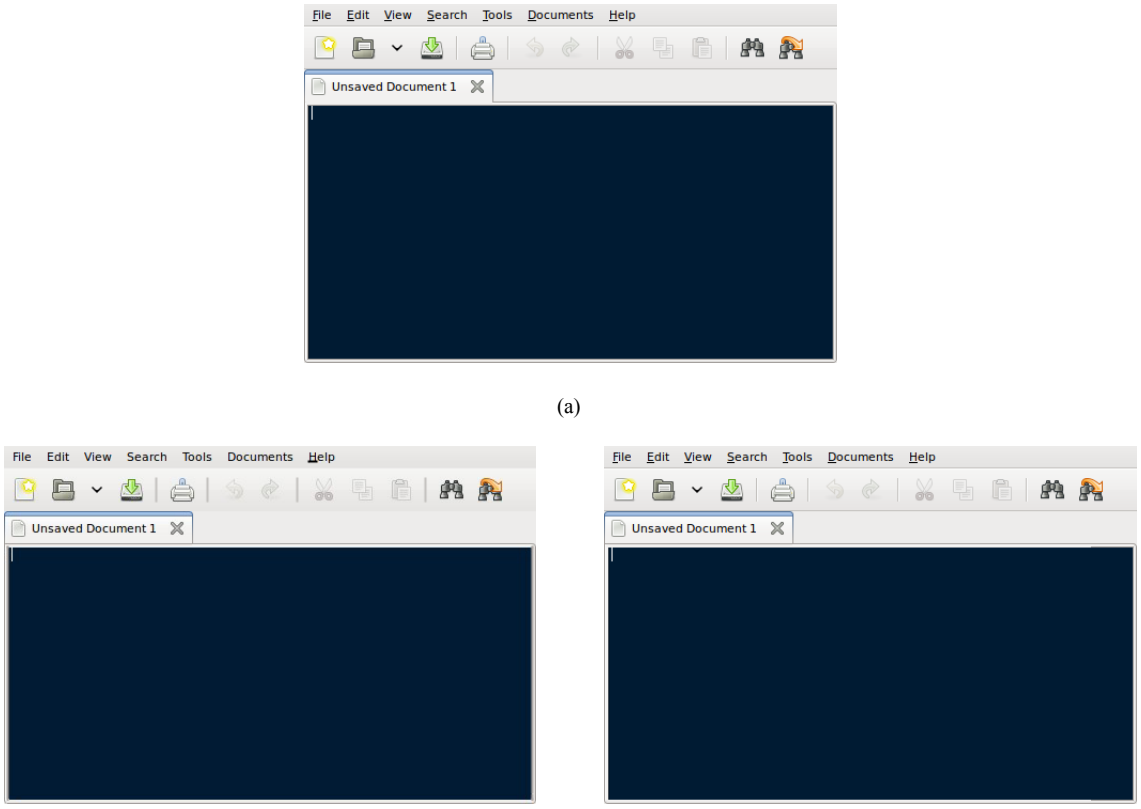
The complete framework was assessed by carrying out two experiments, related to text editing and www browsing.

The gEdit text editing experiment considers 5 users, each of which typing for 5 minutes the text corresponding to the beginning of Plato's Republica.

In order to investigate the case of web browsing, Epiphany was run by 5 users, each of which performing the following actions: (1) load Google page, (2) type "Wikipedia mobile", hit enter and wait for the page to be load, (3) click the Wikipedia mobile link and wait for the Wikipedia mobile page to be loaded, (4) type "chocolate" in the search area, hit enter and wait for the searched result page to be displayed, (5) click the link "bitter" and wait for the new page to load, (6) click the "Bookmark" menu item, select the google.news link, and wait for the page to load, (7) click the home icon, and wait for the www.debian.org home page to load, (8) scroll down, (9) click the "File" menu item and select "Quit".

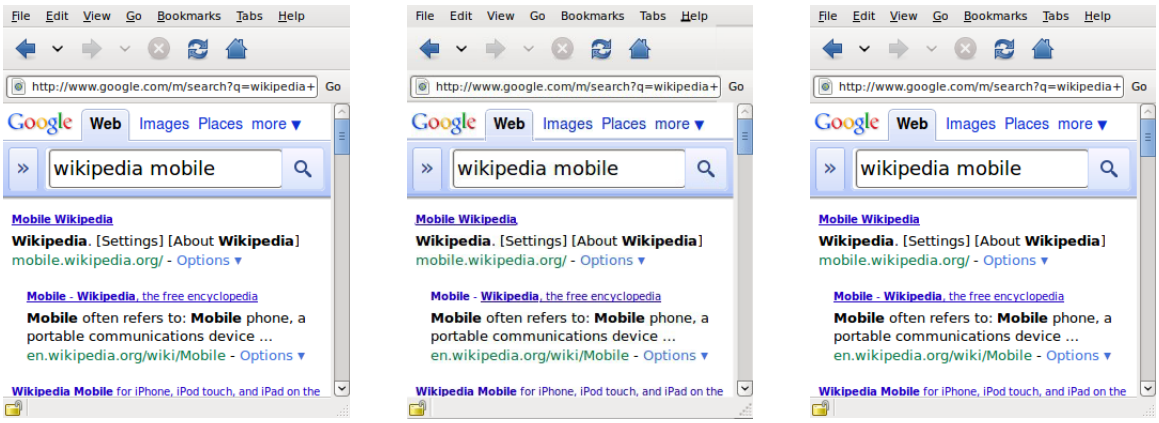
4.2.1. Visual quality

Figures 4.1 and 4.2 illustrate the quality of the MPEG-4 converted content, for the two above mentioned experiments. No illustration has been done for VNC, VNC-HEXTILE and RDP, as their server visual content is kept unchanged during the transmission and displaying; the visual content generated by the MASC-BiFS is identical to the one generated by the basic BiFS. Figures 4.1 and 4.2 also show that although some differences are induced by the MPEG conversion mechanism (*e.g.* in the text editing case, the lines separating the icons are different) they are practically unnoticeable and do not decrease the user experience.



(a) (b) (c)

Figure 4.1. Illustration of the text editing application run on the server (a) and displayed on the mobile thin client, after its conversion into BiFS / MASC-BiFS (b) and MASC-LASeR (c)



(a) (b) (c)

Figure 4.2. Illustration of the www browsing application run on the server (a) and displayed on the mobile thin client, after its conversion into BiFS / MASC-BiFS (b) and MASC-LASeR (c)

The objective evaluations considered two types of measures, Table 4.1: (1) *pixel difference based measures* (PSNR - peak signal to noise ratio, AAD - absolute average difference, and IF - image fidelity) and (2) *correlation based measures* (CQ - correlation quality, SC - structural content, NCC - normalized cross-correlation, and SSIM – structural similarity). The identity between two images

is expressed by the ideal values for these measures (PSNR → ∞, AAD = 0, IF =1, CQ = SC = NCC = SSIM = 1). Note that although no objective quality measure can guarantee the quality perceived by the human observer, they are commonly in use in image processing, [Baroncini, 2012], [Rahmoune, 2006], [Skodras, 2001], [Petrazzuoli, 2010], [Shiang, 2007], [De Simone, 2011].

For the two experiments, in order to assess the visual quality, the rendered visual content corresponding to each and every scene update is converted into pixel maps and is subsequently saved in the ppm format on both server and client sides (thus obtaining pairs of images on which the objective measures are computed). In the case of the text editing experiment, one scene update is generated for each character typed by a user. Consequently, the number of images generated by each user in 5 minutes depends on his/her typing speed; in our experiments, we recorded 652, 827, 753, 694 and 798 characters for the five users, respectively. The related values presented in Table 4.1 (the gEdit columns) are obtained by averaging the visual quality measures obtained for each scene-update and for each user (*i.e.* are computed as average values on 3724 image pairs). As in the case of the www browsing experiments, one scene update is generated for each browsing step, each user generates 9 pairs of images; consequently, the related values presented in Table 4.1 (the Epiphany columns) are computed on 45 image pairs.

In order to offer statistically relevant information about the visual quality assessment, 95% confidence intervals were computed [Fry, 1965], [Walpole, 1989]. For each experiment, each technology and each objective metrics, Table 4.1 presents the average value and the associated 95% error; hence, the corresponding 95% confidence intervals are given by $(average - error\,;\, average + error)$.

In Table 4.1, the PSNR average values (in dB) are approximated to the closest integer, the AAD, IF, CQ, SC and NCC average values are presented with 0.001 precision while a 0.000001 precision was chosen for the average value of SSIM. One more decimal digit was added in each case for the error presentation. Table 4.1 shows that, with singular exceptions (the PSNR and the SSIM values in the case of the Epiphany), the average values become statistical relevant even without considering their confidence limits the 95% estimation error is lower than the precision to which the average values were filled-in in Table 4.1[3].

The values corresponding to MASC-BiFS are identical to the basic BiFS ones. As the VNC, VNC-HEXTILE and RDP do not alter the visual quality, they result in ideal values for the considered measures.

All the values in Table 4.1 demonstrate the visual quality of the MPEG-4 converted content (in the sense of the above-mentioned reference limits for each and every investigated quality metrics). This result is very interesting, as we considered measures designed for natural images and not for heterogeneous visual content, combining text, graphics, icons, and images. This particularity in

[3] When computing the confidence intervals, the correlation between the images corresponding to successive scene updates was neglected; however, because of the very small variance of the values corresponding to each and every quality metric, the practical relevance of the results is not affected by this approximation.

the content can justify some apparently contradictory values in Table 4.1; for instance, in the case of the LASeR conversion of the gEdit, the best PSNR was obtained (42dB) but the related CQ is very low (0.702). When the content produced by the application is closer to natural images (*e.g.* the www browsing case) these discrepancies fade: for the MASC-LASeR conversion, PSNR = 40 dB and CQ = 0.953.

Table 4.1. Visual quality evaluation for X to MPEG (BiFS, MASC-BiFS and MASC-LASeR) conversion

	text editor (gEdit)				www browser (Epiphany)			
	BiFS / MASC-BiFS		MASC-LASeR		BiFS / MASC-BiFS		MASC-LASeR	
	average	*error*	*average*	*error*	*average*	*error*	*average*	*error*
PSNR (dB)	30	0.0	42	0.0	32	1.2	40	1.4
AAD	0.003	0.0000	0	0.0000	0.002	0.0008	0.005	0.0004
IF	0.998	0.0000	0.999	0.0000	0.999	0.0009	0.999	0.0001
CQ	0.929	0.0000	0.702	0.0000	0.974	0.0006	0.953	0.0003
SC	0.995	0.0000	1	0.0000	0.997	0.0005	1.009	0.0007
NCC	1	0.0000	0.999	0.0000	1	0.0004	0.995	0.0041
SSIM	0.999980	0.0000000	0.999999	0.0000000	0.999956	0.0000132	0.999992	0.0000031

4.2.2. Down-link bandwidth consumption

After the scene initialization, information is sent through the network downlink for each and every scene-update, be it initiated by the user (*e.g.* typing a letter or clicking) or by the server (*e.g.* a screen refresh). In the former case, the amount of traffic on downlink depends on the particular action they take (*e.g.* typing a letter will generate less traffic than clicking a menu item). In the latter case, the amount of traffic on downlink is random, depending on the server status and X application behavior.

For the text editing experiment, the values (in KBytes) of the bandwidth required by the corresponding cumulative downlink traffic, averaged over the 5 users, are plotted as a function of time (indexed in minutes) in Figure 4.3 (the value "0" on the abscissa refers to the scene initialization). Note that in this experiment, the number of scene updates varies with the scene updates generated by each user, *i.e.* with the number of letters they actually typed in each time interval (*e.g.*, after 5 minutes, 652, 827, 753, 694 and 798, respectively).

The www browsing experiment is illustrated in Figure 4.4, where the values (in KBytes) of the cumulative network traffic, averaged over the 5 users, are plotted (as a function of the 9 steps).

Note that this time the amount of traffic generated by each user is quite the same (each user generating the same updates) and small differences occurred only because of the server initiated downlink traffic.

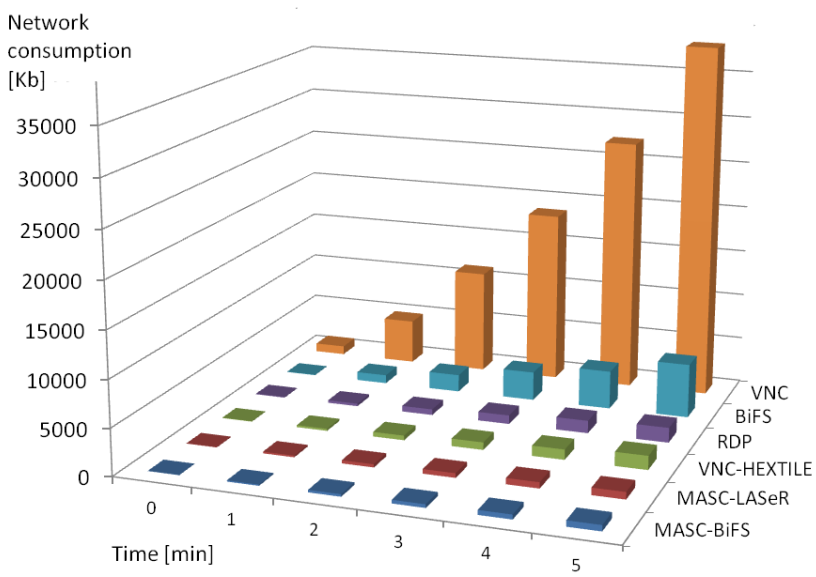


Figure 4.3. Average bandwidth consumption (in KBytes) for text editing (gEdit), as a function of time

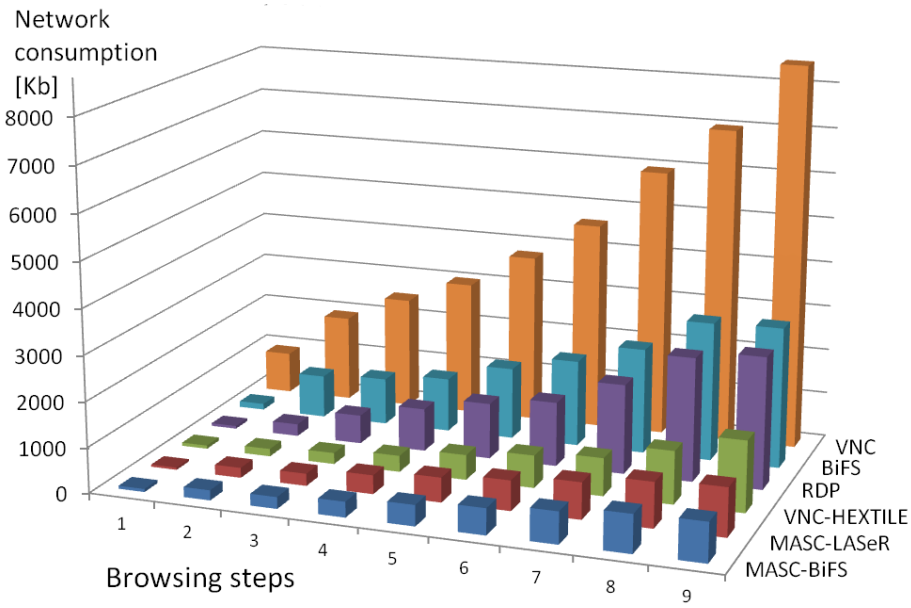


Figure 4.4. Average bandwidth consumption (in KBytes) for www browsing (Epiphany), as a function of the browsing step

Figures 4.3 and 4.4 establish that for the two considered applications, MASC-BiFS is the best solution. In the text editing scenario, it outperforms MASC-LASeR, VNC-HEXTILE, RDP, basic BiFS and VNC by factors of 1.2, 2.3, 2.5, 9.3 and 60, respectively. When considering the www browsing, the MASC-BiFS gain over its competitors ranges from 1.2 to 10.

These compression gains are mainly due to two key factors the MASC-BiFS solution features. First, the visual content sent from the server to the client is no longer considered as a sequence of raw images (*i.e.* pixels) but as a collection of multimedia contents, semantically structured according to their types. This way, each type of content can be compressed with its optimal encoding mechanism. Secondly, the developed scene-graph management mechanism eliminates the need for the retransmission of the visual content that was already sent to the client. Although the application periodically regenerates the same visual content (*e.g.* icons, user actions like "mouse over", *etc.*), the network will no longer be overcharged accordingly. By comparing the results concerning the MASC-BiFS to those corresponding to the BiFS, information about the practical impact of exploiting the semantic information in the scene management is obtained.

The additional down-link traffic generated when the WiFI network connection is lost was also assessed. In both text editing and www browsing experiments, the network connection lost is simulated by switching off the Wi-Fi access point. After 5 seconds, the Wi-Fi access point is switched on again and a new connection with the server is established. The server sends to the terminal the actual status of the application, thus overcharging the overall network traffic.

In the text editing scenario, for each user, we simulated a connection lost at each minute during the 5 minutes of experiments. In the www browsing experiment, the connection lost occurred after each browsing step (so, a total of 9 errors for each user).

The average (over the 5 users) network traffic overcharge induced by reconnection (expressed in kB) is reported in Tables 4.2 and 4.3.

The network overcharge depends not only on the user event type but also on the complexity of the scene. On the one hand, for text editing, the user basically performs the same event which generates a similar type of scene update (a key is stroked and subsequently displayed). In such a case (see Table 4.2) the network overcharge grows with the complexity of the scene (the larger the time, the more complex the scene). On the other hand, for the www browsing experiment, the user events are of different types (mouse click, typing) and generate different types of scene updates; consequently, the network overcharge is not an increasing function of time (browsing step).

The last columns in Table 4.2 and 4.3 show that the MASC-LASeR and MASC-BiFS solutions require the minimum network overcharge for reconnection with the server. MASC-LASER outperforms MASC-BiFS for text editing experiment. This is not the case in www browsing experiment, where the tendency is opposite.

Table 4.2. Average overcharge traffic (in KB) induced by network disconnection, for text editing. On the rows: the mobile thin client technologies; on the columns: the disconnection time (expressed in minutes).

	0	1	2	3	4	5	average
MASC-BiFS	27	20	33	40	44	55	**37**
MASC-LASeR	35	20	26	29	33	37	**30**
VNC-HEXTILE	50	63	104	111	138	126	**99**
RDP	59	100	183	223	263	251	**180**
BiFS	46	147	242	336	369	511	**276**
VNC	941	3948	1572	5721	4907	9358	**4408**

Table 4.3. Average overcharge traffic (in KB) induced by network disconnection, for www browsing. On the rows: the mobile thin client technologies; on the columns: the disconnection time (expressed in browsing steps).

	1	2	3	4	5	6	7	8	9	average
MASC-BiFS	67	101	87	86	59	76	88	87	112	**85**
MASC-LASeR	50	99	130	83	55	73	80	76	157	**89**
VNC-HEXTILE	73	134	184	130	85	123	119	152	159	**129**
RDP	61	193	371	344	191	192	379	277	259	**252**
BiFS	135	437	353	300	204	251	296	322	404	**300**
VNC	920	1693	1515	1510	1211	2323	1752	2180	2813	**1769**

4.2.3. User interaction efficiency

As previously mentioned, the MPEG-4 BiFS standard makes provisions for two different ways of transmitting the user interactivity through the up-link: AJAX HttpRequest and *ServerCommand*. Consequently, in this section, the BiFS and MASC-BiFS will be considered in two different cases, according to their ways of exploiting the up-link.

In our study, we considered the two most frequent user events: keyboard strokes and mouse (pointing device) clicks.

The size of traffic generated through the up-link channel, as measured for each solution, is represented in Table 4.4. These values depend on the technology but are independent with respect to the particular event (typing E or A generates the same traffic, right click generates the same traffic as the left click, *etc.*) and to the network conditions.

Table 4.4 also provides information about the network round-trip times, *i.e.* the time elapsed between the moment when the user interactivity actually takes place and the moment when the updated scene-graph is displayed. As similar interaction mechanisms are obtained for keyboard strokes and mouse clicks, the related round-trip times are to be equal. However, these values

slightly depend on the network conditions. The values presented in Table 4.4 are obtained as average values over all the users and all the 3894 events they generated: 3724 characters for gEdit, 125 characters for Epiphany (5 users typing "Wikipedia mobile" and "chocolate") and 45 clicks. The corresponding 95% confidence intervals featured errors lower than 1ms.

Table 4.4. The size of the traffic generated through the back channel by elementary user events

	traffic (bytes)		roundtrip-time (ms)
	keyboard stroke	mouse click	keyboard stroke / mouse click
VNC / VNC-HEXTILE	586	586	80
RDP	186	618	130
BiFS / MASC-BiFS /MASC-LASeR [AJAX HTTPRequest]	564	581	20
BiFS / MASC-BiFS [ServerCommand – TCP]	72	82	18
BiFS / MASC-BiFS [ServerCommand – UDP]	46	56	18

Table 4.4 shows that BiFS / MASC-BiFS solution considering the *ServerCommand* using UDP requires the lowest bandwidth, reaching 46 bytes (*i.e.* an up-link bandwidth gain factors from 4 to 12) for a keyboard stroke and 56 bytes (*i.e.* an up-link bandwidth gain factors from 10 to 11) for a mouse click, while keeping the interactivity round-trip times at 18ms. The same minimal round-trip times (18ms) are obtained for BiFS / MASC-BiFS considering the *ServerCommand* using TCP; however, with respect with the VNC/VNC-HEXTILE and RDP, the gains in the up-link bandwidth range now between 2.5 and 8. No clear advantage of the ServerCommand over the AJAX HTTPRequest has been identified by this experiment.

Note: Table 4.4 reports only the values corresponding to the server-side interactivity, the most disturbing solution from the QoE point of view.

4.2.4. CPU activity

The amount of processor power needed to run the remote display client in order to render all the streamed content is assessed in this section. As from this point of view the relevant information is brought by the maximal CPU usage, in this experiment we considered only the www browsing application.

The measurements presented in Figure 4.5 are devoted to the values of the maximum CPU activity when browsing the www. It can be noticed that the remote display solutions that use raw pixel representation of the images (BiFS and VNC) produce less CPU activity than the rest (MASC-BiFS, MASC-LASeR and VNC-HEXTILE). However, it can be seen that the MASC-BiFS solutions does

not exceed the maximal CPU activity of 58% (browsing step 7), compared with the LASeR reaching 68% (browsing step 9) and VNC-HEXTILE 95% (browsing step 7) of the total available computational resources on the device. This makes the MASC-BiFS solution even more appropriate for thin clients.

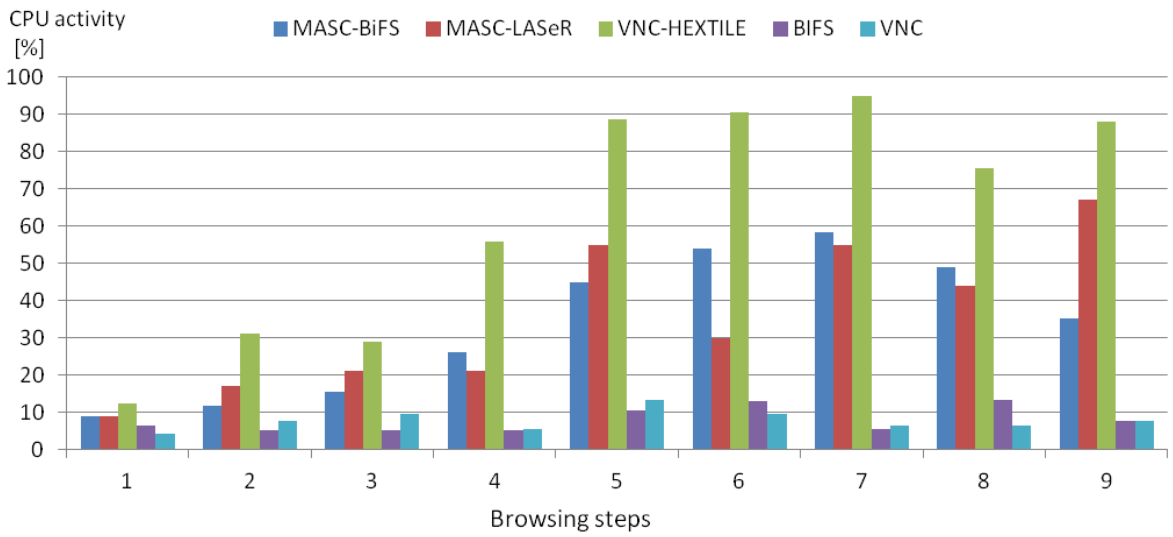


Figure 4.5. The average maximum CPU consumption (in %) while browsing, as a function of the browsing step

Note that the RDP case is not represented in the Figure 4.5, as it is a solution integrated into the thin client Windows mobile OS and its accurate measurement is practically impossible to obtain. However, the experiments we carried out pointed to the fact that the RDP is the lightest solution.

4.3 Discussions

This section details the practical relevance of the Semantic Controller and Pruner.

4.3.1. Semantic Controller performance

The experiments reported in Section 4.2 considered that the Semantic Controller processes png compressed images (compression level 9, and 24 bit depth). This section considers alternative image management solutions:

- jpeg compression with parameters: visual quality 90% and 75% (denoted by MASC-BiFS jpeg 90 and MASC-BiFS jpeg 75, respectively);
- raw (uncompressed) images (denoted by MASC-BiFS RAW).

For benchmarking this block (see Figures 4.6 and 4.7), the same two experiments of text editing and www browsing are considered. The Semantic Controller was firstly disabled (not considered in the architecture) then enabled It is thus established that the Semantic Controller reduces the network traffic by about 50%, for a same compression type.

Note: the MASC-BiFS RAW solution with activated Semantic Controller corresponds to the BiFS solution in Section 4.2.

A particular behaviour concerning image compression should also be noted: for the images generated by these two types of applications, the png mechanism results in better compression rates than the jpeg mechanism. This is a consequence of both the size and the content of such images.

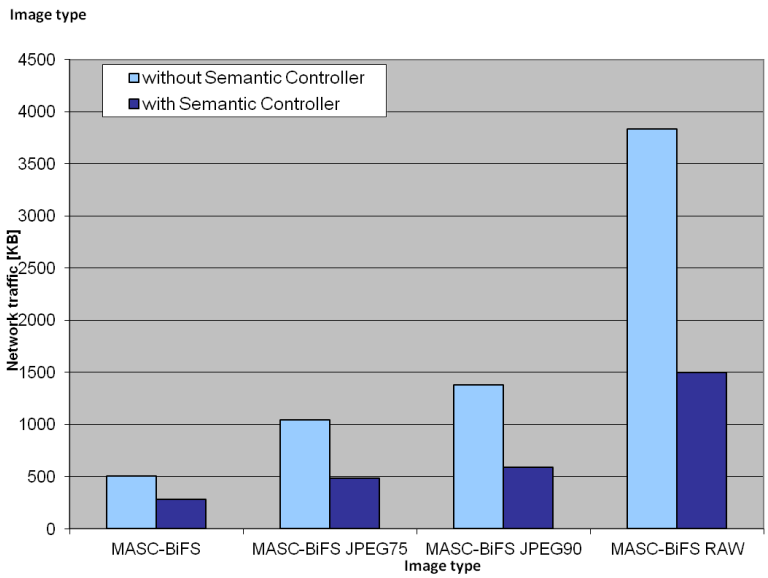


Figure 4.6. Performance of the Semantic Controller block: total bandwidth consumption in the case of text editing over image type (encoding used for the images in the scene-graph)

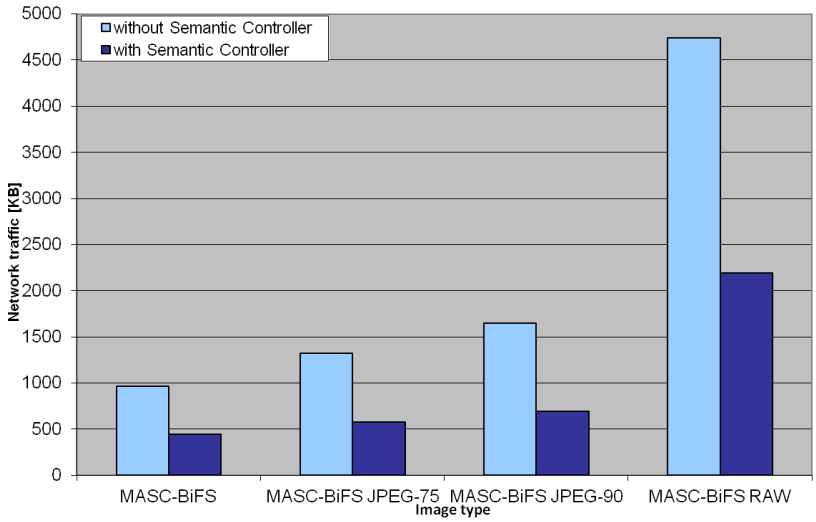


Figure 4.7. Performance of the Semantic Controller block: total bandwidth consumption in the case of www browsing (executing the 9 steps) over image type (encoding used for the images in the scene-graph)

4.3.2. Pruner performance

The addition of new nodes to a scene generally results in increased CPU activity, assuming the scene nodes list is continuously increasing. So deleting a node should equally decrease the CPU activity. Hence, the network consumption amelioration granted by the Semantic Controller is obtained at the expense of computational activity.

In order to stabilize the computational activity at the client side, we focused our attention on the reduction of the node count of the scene at any one moment in time. This action resulted in creation of the *Pruner,* explained in Section 3.6.1.

In order to benchmark the performances of the *Pruner* we used the architecture in two modes: (1) without the Pruner and (2) with the Pruner, by using the same experiments of text editing and www browsing.

The results obtained in the first experiment, the text editing, are plotted in Figure 4.8, where average maximum CPU activity is represented as a function of time (in seconds). By analyzing the Figure 4.8, we can notice the Pruner reduces the CPU activity by a factor of 5.

The second experiment (the www browsing), see Figure 4.9, shows that the *Pruner* reduces the CPU consumption by a factor of 2.5.

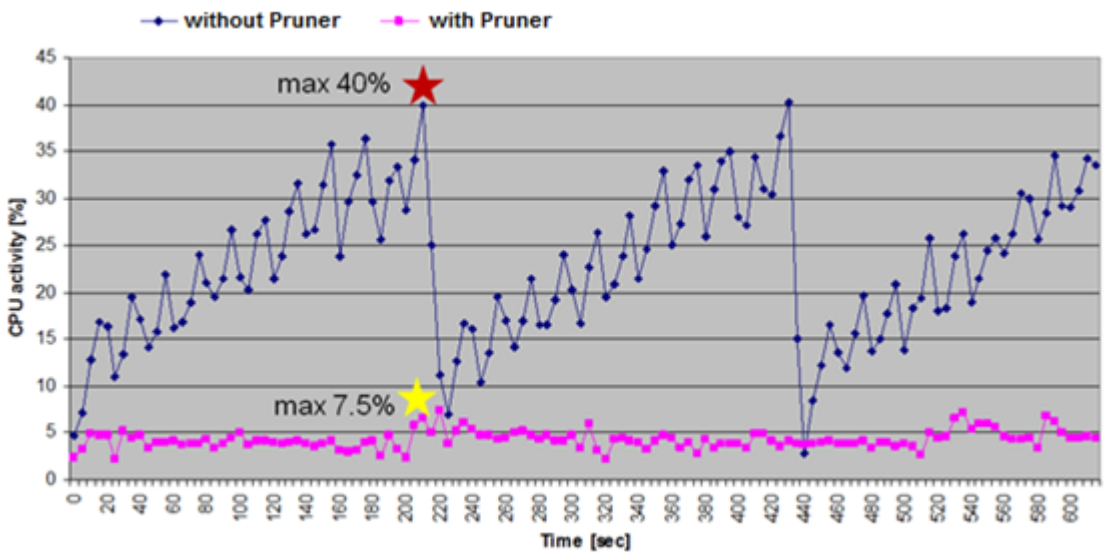


Figure 4.8. Average maximum CPU activity as a function of time expressed in seconds, in the text editing experiment

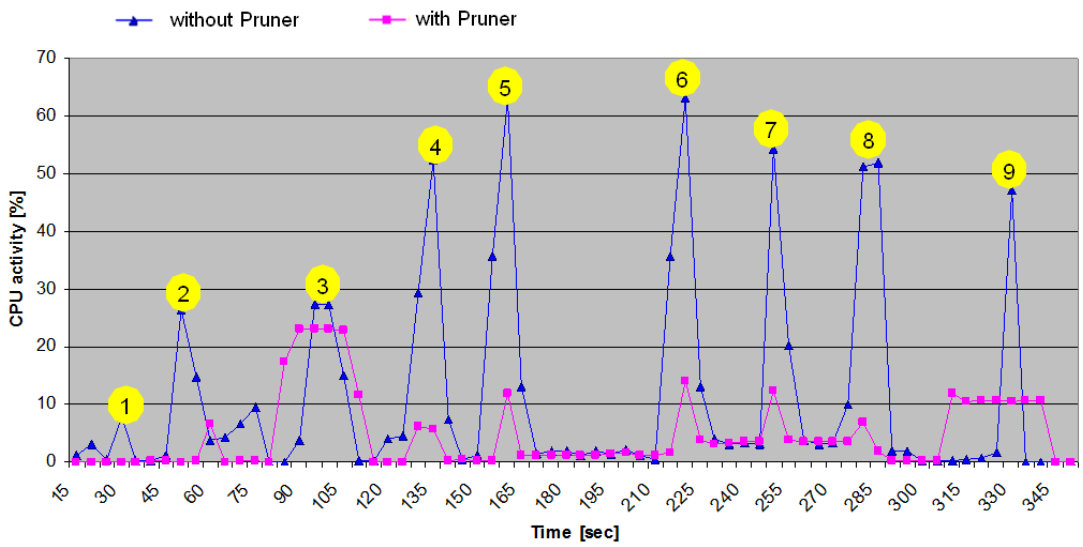


Figure 4.9. Average maximum CPU activity as a function of time expressed in seconds, in the www browsing experiment

Chapter 5. Conclusion and Perspectives

Cette thèse propose la première architecture logicielle pour un système de rendu distant, basée sur (1) la gestion sémantique du contenu multimédia pour assurer une optimisation conjointe de l'utilisation du réseau et des ressources calcul côté terminal et (2) des nouveaux éléments de traitement des données de collaboration directement au niveau du contenu multimédia, pour assurer des solutions normatives en logiciel libre.

MASC est évaluée pour son potentiel industriel dans différents domaines applicatifs, tels que la visualisation des applications dans le nuage (en partenariat avec Prologue et Skypath), la réalisation d'un système collaboratif de décision dans les applications de vidéosurveillance intelligentes (en partenariat avec CASSIDIAN) ou bien encore l'aide au diagnostique dans un environnement virtuel et collaboratif (en partenariat avec Philips HealthCare et Bull).

5.1 Conclusion

To the best of our knowledge, the present thesis advances the first semantic multimedia remote viewer for collaborative mobile thin clients. In this respect, a new open-source, open-standard client-server architecture is specified, designed, implemented and validated. Its main enablers are (see Figure 5.1):

- *On the client-side*:
  - ISO multimedia scene-graph representations extended with elements devoted to the real-time user collaboration; these new scene-graph elements are currently under an ISO standardization process to be achieved by 2013: *ISO/IEC 14496-11: version 2 – Information technology - Coding of audio-visual objects - Part 11: Scene description and application engine*;
  - dynamic compression algorithm for user presence signaling in collaborative environments (patented solution).
- *On the server-side*:
  - real-time scene-graph adaptation (composition, compression, …) algorithm (patent pending);
  - semantic management framework for multimedia scene-graphs (ISI journal publication).

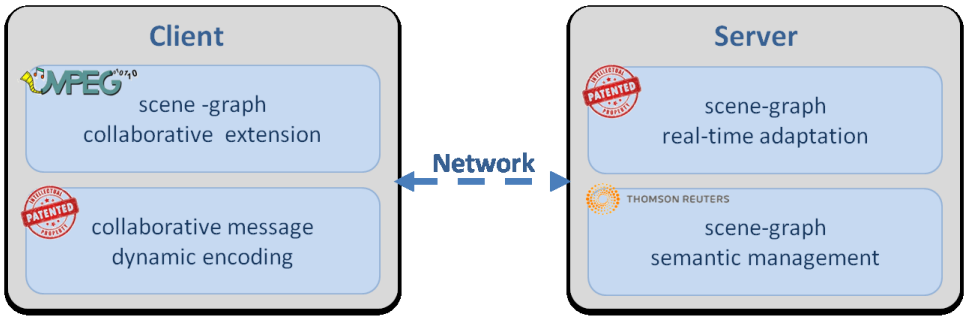


Figure 5.1. Enablers for the advanced collaborative mobile thin client framework

The architecture is implemented as an open-source, open-standard software demonstrator referred to as MASC (Multimedia Adaptive Semantic Collaboration). The MASC benefits are twofold.

On the one hand, from the functional point of view, MASC is the first mobile thin client solution featuring at the client side: (1) full multimedia experience; (2) virtually unlimited collaboration options and (3) cross-standard support.

On the other hand, from the quantitative assessment point of view, MASC outperforms its state-of-the-art competitors (VNC (RFB) and RDP) by featuring: (1) high level visual quality, *e.g.* PSNR values between 30 and 42dB or SSIM values larger than 0.9999; (2) downlink band-width gain factors ranging from 2 to 60; (3) efficient real-time user event management expressed by

network roundtrip-time reduction by factors of 4 to 6 and by up-link bandwidth gain factors from 3 to 10; (4) feasible CPU activity, larger than in the Microsoft RDP case but reduced by a factor of 1.5 with respect to the VNC-HEXTILE.

5.2 Perspectives

The perspectives of our work are connected to demonstrating the generality of the architecture. In this respect, its main blocks are reconsidered and adapted to other applicative frameworks:

- *extension from Linux to Windows applications*
  From a conceptual point of view the Windows applications can be dealt with when assuming some rich multimedia content is available at a given level of the Windows application and when this content can be subsequently listen to, parsed and converted into MPEG-4 BiFS content. From the practical point of view, it can be noted that the RDP (see Section 2.2.5) offers that collection of multimedia content. Hence, in our study it will consider as the level to which the Windows application will be intercepted, see Figure 5.2.
- *extension from MPEG-4 BiFS and LASeR to HTML5 clients*
  From the conceptual point of view, there is no contradiction between MPEG-4 and HTML5 content representation: although differently aggregated, the native content itself is the same (for instance, text, jpg/png image, MPEG-4 AVC video, …). Consequently, from the technical point of view, replacing all the BiFS/LASeR blocks from the architecture in Figure 3.3 with their HTML5 counterparts is expected to be straightforward.
- *beyond MPEG-4 collaboration*
  The architectural enablers provided by our study (the semantic management, the compression algorithms, the collaboration support) are designed at the content level, independent with respect to any application/operating system peculiarity. Consequently, they can be adapted for serving cross-standard collaborative environments, *i.e.* environments in which users are powered with different standard terminals (e.g. MPEG-4 and HTML5). One possible solution in this respect is presented in Figure 5.3.

The industrialization perspectives are open by the fact that the novel solution is based on open-source architecture, Figures 3.3 and 3.4. The way in which the modules are developed and located alleviates the need for the modification of the legacy software (be it OS or application) and allows a straightforward integration into emerging commercial application platforms, with minimal modification on both server and client sides. On the one hand, the server should be updated with the architectural framework while the rest of the applications can be kept unchanged. On the other hand, at the client, only an MPEG-4 player needs to be installed.

Such an approach completely satisfies the requirements of the mobile device switching. Firstly, all administration tasks are to be performed on the server-side: applications (*e.g.* www browsing) can be installed/updated/removed/replaced on the server without changing anything on the

client. Secondly, the terminal independence is ensured by the MPEG-4 ISO standards. For instance, the GPAC framework is already available for use on most thin client terminals (Windows Mobile, Android and Apple iOS) and desktop computers (Windows, Linux and Mac OS). Moreover, it is able to play all types of MPEG multimedia content: audio, 2D, video, 3D BiFS, LASeR, VRML, SVG, 3GPP and so forth. Note that the MASC-BiFS solution is perfectly compliant with the MPEG-4 BiFS standard; however, its development required the adaptation of the GPAC Framework, particularly concerning the collaboration. Thirdly, all components are supported by strong and rising open source communities, thus ensuring their potential evolution.

These three properties appeal to the various industrial players, from telco operators and service providers to third party software editors. The interest towards the architecture advanced in this paper is even broader in perspective, with the advent of cloud computing [Baliga, 2011], and of modern distributed collaborative environments.

The MASC ongoing actions consider evaluation for its potential industrialization in various applicative fields, like application virtualization in the cloud (in partnership with Prologue), collaborative decision making system for video surveillance applications (in partnership with CASSIDIAN) and virtual collaborative environments for medical assistance (in partnership with Philips HealthCare and Bull).

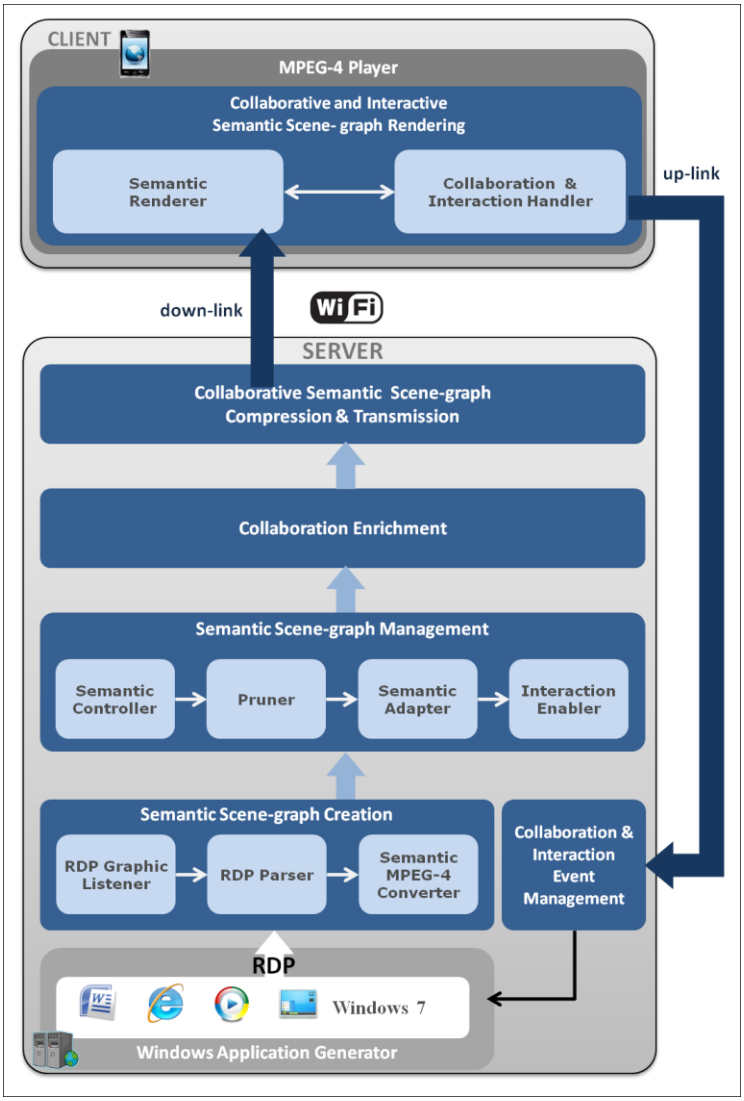


Figure 5.2. Extension from Linux to Windows applications

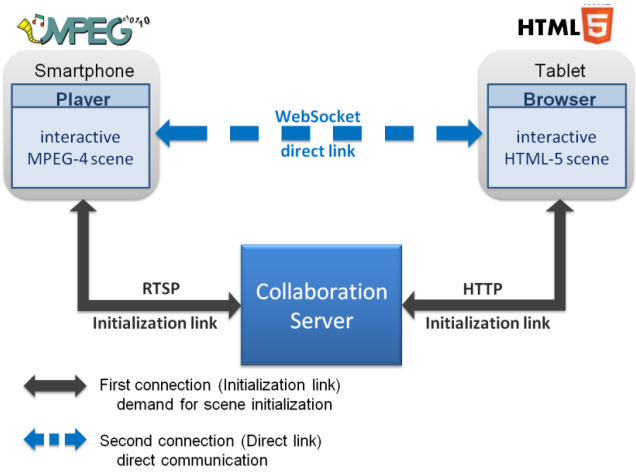


Figure 5.3. Beyond MPEG collaboration – cross standard collaboration

List of abbreviations

3GPP	Third Generation Partnership Project
AAD	Absolute Average Difference
AJAX HttpRequest	Asynchronous Javascript And XML HyperText Transfer Protocol Request
AVC	Advanced Video Coding
BiFS	Binary Format for Scene
BiM	Binary MPEG
CPU	Central Processing Unit
CQ	Correlation Quality
DASH	Dynamic Adaptive Streaming over HTTP
DEMUX	Demultiplexing
ECMA	European Computer Manufacturer Association
FLV	FLash Video
GDI	Graphical Device Interface
GSM	Global System for Mobile
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol
IF	Image Fidelity
I/O	Input / Output
iOS	iPhone Operating System
ISO	International Organization for Standardization
LASeR	Lightweight Application Scene Representation
Mac OS	Apple Operating System
MASC	Multimedia Adaptation Semantic Collaboration
MPEG	Moving Picture Expert Group
MUX	Multiplexing
MMT	MPEG Multimedia Transport
NCC	Normalized Cross-Correlation
OS	Operating System
PC	Personal Computer
PDA	Personal Digital Assistant
png	Portable Network Graphics
ppm	Portable Pixel Map
PSNR	Peak Signal-to-Noise Ratio
QoE	Quality of Experience
RDP	Remote Desktop Protocol
RFB	Remote FrameBuffer

RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SAF	Simple Aggregation Format
SC	Structural Content
SMIL	Synchronized Multimedia Integration Language
SSIM	Structural SIMilarity
SVG	Scalable Vector Graphics
SWF	ShockWave Flash
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine
VNC	Virtual Network Computing
VRML	Virtual Reality Modeling Language
Wi-Fi	Wireless Fidelity
xHTML	eXtensible HyperText Markup Language
XML	eXtensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

References

[Adobe, 2005] Adobe, www.adobe.com, 2012

[AVC, 2012] JTC1 SC 29 ISO/IEC 14496-10 (2012), *"Information technology -- Coding of audio-visual objects -- Part 10: Advanced Video Coding"*, http://standards.iso.org/ittf/PubliclyAvailableStandards/c061490_ISO_IEC_14496-10_2012.zip, 2012

[Asadi, 2005] M.K. Asadi, J.-C Dufourd, "Context-Aware Semantic Adaptation of Multimedia Presentations", IEEE International Conference on Multimedia and Expo, Amsterdam, Holland, 6-8 july 2005.

[Baliga, 2011] J. Baliga, R.W.A. Ayre, K. Hinton, R.S. Tucker, "Green Cloud Computing: Balancing Energy in Processing, Storage, and Transport", Proceedings of the IEEE, Vol. 99, No.1, January 2011

[Baroncini, 2010] Vittorio Baroncini, Jens-Rainer Ohm, Gary J. Sullivan, "Report of Subjective Test Results of Responses to the Joint Call for Proposals (CfP) on Video Coding Technology for High Efficiency Video Coding (HEVC)", ISO/IEC JTC1/SC29/WG11 MPEG2010/N11275 Dresden DE, April 2010

[Basso, 2002] A. Basso, Y.-J. Kim, Z. Jiang, ,"Performance evaluation of MPEG-4 video over realistic EDGE wireless networks", The 5th International Symposium on Wireless Personal Multimedia Communications, Volume: 3 Page(s): 1118- 1122 vol.3, 2002

[Battista, 1999] S. Battista, F. Casalino, C. Lande, "*MPEG-4: A Multimedia Standard for the Third Millennium, Part 1"*, IEEE MultiMedia, pp: 74 - 83, January–March 1999

[Battista, 2000] S. Battista, F. Casalino, C. Lande, "*MPEG-4: A Multimedia Standard for the Third Millennium, Part 2",* IEEE MultiMedia, pp:76-84, January–March 2000

[Beg, 2002] Beg, M.S., A. Muslim, Y. C. Chang , T. F. Tang , "*Performance evaluation of error resilient tools for MPEG-4 video transmission over a mobile channel*", Conference on Personal Wireless Communications, 15-17 Dec. 2002

[BiFS, 2012] J. Marshall, Y. Lim, "*Request for 2nd edition of ISO/IEC 14496 Part 11: Scene Description and Application Engine*", ISO/IEC JTC1/SC29/WG11 N12549, Geneva, CH, May 2012

[Blizzard, 2012] Blizzard, "*Game: World Of Warcraft*", http://us.battle.net/wow/en/, 2012

[Bruno, 2006] E.J. Bruno, "Ajax: Asynchronous JavaScript and XML", *Dr. Dobb's Journal*, v 31, n 2, p 32-35, February 2006,

[BiFS, 2005] TC/JTC/SC29 ISO/IEC 14496-11:2005, "*Information technology -- Coding of audio-visual objects -- Part 11: Scene description and application engine",* http://www.iso.org, 2012

[Carroll, 2010] A. Carroll, G. Heiser, *"An Analysis of Power Consumption in a Smartphone",* USENIX 2010, June 22–25, 2010

[Cisco, 2012] Cisco Analysis, 2012 "*Cisco Global Cloud Index: Forecast and Methodology, 2011–2016*", http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns1175/Cloud_Index_White_Paper.html, 2012

[Concolato, 2008] C. Concolato, J. Le Feuvre, J.-C. Moissinac, "*Design of an Efficient Scalable Vector Graphics Player for Constrained Devices*", IEEE Transactions on Consumer Electronics, Vol. 54, No. 2, May 2008

[DASH, 2011] MPEG Dynamic Adaptive Streaming over HTTP (DASH), http://mpeg.chiariglione.org/working_documents/mpeg-dash/MPEG-DASH-Tutorial.pdf, 2012

[De Simone, 2011] Francesca De Simone, Matteo Naccari, Marco Tagliasacchi, Frederic Dufaux, Stefano Tubaro, Touradj Ebrahimi, "Subjective Quality Assessment of H.264/AVC Video Streaming with Packet Losses", EURASIP Journal on Image and Video Processing, Volume 2011

[Dufourd, 2005] J-C. Dufourd, O. Avaro, C. Concolato, "An MPEG Standard for Rich Media Services", IEEE MultiMedia, pp: 60-68, October–December 2005

[Duta, 2007] S. Duta, M. Mitrea, F. Prêteux. "Compressed versus uncompressed domain video watermarking", Proc. SPIE, Vol. 6700, p. 67000A, August 2007

[Epiphany, 2012] Open-source community, "*Epiphany web browser for the GNOME desktop environment*", http://projects.gnome.org/epiphany/, 2012

[Facebook, 2012] Facebook, "*Facebook statistics*", www.facebook.com, 2012

[FaceTime, 2012] Video calling, http://www.apple.com/ios/facetime/, 2012

[Fry, 1965] T.C. Fry, "Probability and Its Engineering Use", D van Nostrand, 1965

[Galluccio, 2005] L. Galluccio, "*Transmission of adaptive MPEG video over time-varying wireless channels: modeling and performance evaluation*", IEEE Transactions on Wireless Communications, Volume: 4 , Issue: 6 Page(s): 2777- 2788, November, 2005

[Gartner,2012] B. Butler, (2012), "Gartner: Cloud to grow 20% this year to $109B market", http://www.networkworld.com/news/2012/ 091812-gartner-cloud-market-262546.html, 2012

[gEdit, 2012] Open-source community, "*gEdit official text editor for the GNOME desktop environment*", http://projects.gnome.org/gedit/, 2012

[GPAC, 2012] J. Le Feuvre, C. Concolato, R. Bouqueau, "*GPAC – Open Source Multimedia Framework*", http://gpac.sourceforge.net/index.php, 2012

[GSMA, 2012] O. Malik, "*Internet of things will have 24 billion devices by 2020*", http://gigaom.com/cloud/internet-of-things-will-have-24-billion-devices-by-2020/, 2012

[HTTP, 1999] R. Fielding, UC Irvine, J. Gettys, Compaq/W3C, J. Mogul, Compaq, H. Frystyk, W3C/MIT, L. Masinter, Xerox, P. Leach, Microsoft, T. Berners-Lee, (June 1999), "*Hypertext Transfer Protocol specification*", RFC2616, http://www.http-compression.com/rfc2616.txt, 2012

[Intel 2012] Intel corporation, "*What Happens In An Internet Minute?*" http://www.intel.com/content/www/us/en/communications/internet-minute-infographic.html, 2012

[Izquierdo, 2003] E. Izquierdo, J.R. Casas, R. Leonardi, P. Migliorati, Noel E. O'Connor, I. Kompatsiaris, M. G. Strintzis, "Advanced content-based semantic scene analysis and information retrieval: the chema project", Workshop on Image Analysis for Multimedia Interactive Services, 9-11 April 2003, London, UK.

[Java, 2005] J. Gosling, B. Joy, G. Steele, G. Bracha, *"The Java Language Specification",* Third Edition, ADDISON-WESLEY, USA

[JavaScript, 2011] Ecma ECMA-262, "*ECMAScript Language Specification*", Ecma International, Geneva, Swiss, June 2012

[Joveski, 2010] B. Joveski, M. Mitrea, F. Preteux, "MPEG-4 LASeR - based thin client remote viewer", EUVIP2010 - European Workshop on Visual Information Processing, Paris, July 2010

[Joveski, 2011] B. Joveski, P.Simoens, L.Gardenghi, J.Marshall, M. Mitrea, B. Vankeirsbilck, F. Prêteux, B. Dhoed, "Towards a multimedia remote viewer for mobile thin clients", in Proceedings of SPIE, Vol: 7881, 788102 (2011); doi:10.1117/12.876279, 2011, San Francisco, January 2011

[Joveski, 2013] B. Joveski, M. Mitrea, P. Simoens, I.J. Marshall, F. Prêteux, B. Dhoedt, "Semantic multimedia remote display for mobile thin clients", Multimedia Systems, DOI: 10.1007/s00530-013-0304-6, January 2013.

[JPEG, 1996] Network Working Group, working draft (1996), "*Multipurpose Internet Mail Extensions (MIME) Part Five:Conformance Criteria and Examples*", http://tools.ietf.org/html/rfc2049, 2012

[LASeR, 2008] ISO/IEC 14496-20:2008, *"Information technology -- Coding of audio-visual objects -- Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)",* http://standards.iso.org/ittf/PubliclyAvailableStandards/c052454_ISO_IEC_14496-20_2008(E).zip, 2012

[Liu, 2005] Z. Liu, Y. Saifullah, M. Greis, S. Sreemanthula, "*HTTP Compression Technique*", Wireless Communications and Networking Conference, 2005 IEEE, pp. 2495 - 2500 Vol. 4, New Orleans, USA, March 13-17 2005

[Live555, 2012] Open source support, (2012), "*Live555 Streaming Media"*, source code libraries, www.live555.com, 2012

[Market, 2012] Net marketshare, "O*perating-system-market-share*", www.netmarketshare.com, 2012

[Marshall, 2012] I.J. Marshall, M. Mitrea, B. Joveski, L. Gardenghi, F. Prêteux, *"Codage de données sans perte pour communication bidirectionnelle",* French patent request # 1151727, March 2011 ; international patent request PCT/FR2012/050428, March 2012

[Mitrea, 2009] M. Mitrea, P. Simoens, B. Joveski, I. J. Marshall, A. Tanguengayte, F. Preteux, B. Dhoed, "BiFS based approaches to remote display for mobile thin clients", in Proceedings of SPIE, Vol: 7444, 74440F (2009); doi:10.1117/12.828152, San Diego, August 2009

[Mitrea, 2012] M. Mitrea, B. Joveski, L. Gardenghi, I.J. Marshall, F. Prêteux, *"Procédés et appareils de production et de traitement de représentations des scènes multimedia",* French patent request # 1153387, April 2011 ; international patent request PCT/FR2012/050849, April 2012

[MMT, 2010] MPEG Media Transport (MMT), http://mpeg.chiariglione.org/working_documents/mpeg-h/mmt/mmt_co.zip, 2012

[MPEG-2, 2007] JTC1 SC 29 ISO/IEC 13818-1 (2007), "*Information technology -- Generic coding of moving pictures and associated audio information: Systems",* http://www.iso.org, 2012

[Nye, 1990] A. Nye (1990), "*Xlib Reference Manual*", Version 11 Vol. 2, O'Reilly & Associates, Inc., http://www.x.org

[Petrazzuoli, 2010] Giovanni Petrazzuoli, Marco Cagnazzo, Beatrice Pesquet-Popescu, "High order motion interpolation for side infor-mation improvement in DVC", IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP, April 2010

[PNG, 2004] ISO/IEC 15948, "*Information technology -- Computer graphics and image processing -- Portable Network Graphics (PNG): Functional specification"*, http://www.iso.org, 2004

[Qiao, 2006] Y. Qiao, Q. Hu, G. Qian, S. Luo, and W. L. Nowinski, "Thresholding based on variance and intensity contrast," Pattern Recognition, vol. 40, no. 2, pp. 596 - 608, July 2006.

[Rahmoune, 2006] Adel Rahmoune, Pierre Vandergheynst, Pascal Frossard, "Flexible motion-adaptive video coding with redundant expansions", IEEE Transactions on Circuits and Systems for Video Technology, vol. 16, no. 2, February 2006

[RemoteFX, 2011] Microsoft Corporation, (2011), "*Microsoft RemoteFX for Virtual Desktop Infrastructure: Architectural Overview*", Microsoft Corporation, January 2011

[Richardson, 2010] T. Richardson, (2010), *"The RFB Protocol*", Version 3.8, RealVNC Ltd, 2010

[Schlosser, 2007] D. Schlosser, A. Binzenhofer, B. Staehle, "*Performance Comparison of Windows-based Thin-Client Architectures*", 2007 Australasian Telecommunication Networks and Applications Conference, Christchurch, New Zealand, December 2 – 5 2007

[Shiang, 2007] Hsien-Po Shiang, Mihaela van der Schaar, "Multi-User Video Streaming Over Multi-Hop Wireless Networks: A Distributed, Cross-Layer Approach Based on Priority Queuing", IEEE Journal on Selected Areas in Communications, vol. 25, no. 4, May 2007

[Simoens, 2008] P. Simoens, P. Praet, B. Vankeirsbilck, J. De Wachter, L. Deboosere, F. De Turck, B. Dhoedt, P. Demeester, "*Design and implementation of a hybrid remote display protocol to optimize multimedia experience on thin client devices*", ATNAC 2008, December 7-10 2008

[Simoens, 2012] P. Simoens, B. Joveski, L. Gardenghi, I.J. Marshall, B. Vankeirsbilck, M. Mitrea, F. Prêteux, F. De Turck, B. Dhoedt, "Optimized mobile thin clients through a MPEG-4 BiFS semantic remote display framework", Springer –Multimedia Tools and Applications, DOI: 10.1007/s11042-011-0849-3, Vol:61/2, pages: 447 – 470, November 2012.

[Skodras, 2001] Athanassios Skodras, Charilaos Christopoulos, Touradj Ebrahimi, "The JPEG 2000 still Image compression stand-ard", IEEE Signal Processing Magazine, September 2001

[Skype, 2012] Instant messanger, http://www.skype.com, 2012

[Song, 2011] J. Song, B.-D. Lee, "Mobile Rich Media Technologies: Current Status and Future Directions", KSII Transactions on Internet and Information Systems, Vol. 5, No. 2, February 2011

[Valve, 2012] Valve Corporation, "*Games: Counter Strike*", Officaial release by Valve Corporation, August 2012

[SMIL/SVG, 2011] W3C Recommendation, "*SVG 1.1 (Second edition)*", http://www.w3.org/TR/SVG/intro.html, 2012

[TimedText, 2010] W3C Recommendation, "*Timed Text Markup Language (TTML) 1.0*" http://www.w3.org/TR/2010/REC-ttaf1-dfxp-20101118/, 2012

[Walpole, 1980] R.E. Walpole and R.H. Myers, "Probability and Statistics for Engineers and Scientists", 4th ed., MacMillan Publishing, NY, 1989

[Wikipedia 2012] Wikipedia, "*Wikipedia statistics*", www.wikipedia.com, 2012

[xHTML, 2009] W3C working draft, "*A vocabulary and associated APIs for HTML and XHTML*", http://www.w3.org/TR/html5/, 2012

[XRdp, 2009] XRDP, "Open source project for Remote Desktop Protocol for Linux", version 0.4.2, http://sourceforge.net/projects/xrdp/files/xrdp/0.4.2/, 2012

[Youtube, 2012] Youtube, "*Youtube statistics*", www.youtube.com, 2012

[Yang, 2002] S. Jae Yang, Jason Nieh, Matt Selsky, Nikhil Tiwari, "*The Performance of Remote Display Mechanisms for Thin-Client Computing*", Proceedings of the 2002 USENIX Annual Technical Conference, Monterey, California, USA, June 10-15, 2002

Appendix I

M. Mitrea, B. Joveski, L. Gardenghi, I.J. Marshall, F. Prêteux, *"Procédés et appareils de production et de traitement de représentations des scènes multimedia"*, French patent request # 1153387, April 2011 ; international patent request PCT/FR2012/050849, April 2012

Appendix II

I.J. Marshall, M. Mitrea, B. Joveski, L. Gardenghi, F. Prêteux, *"Codage de données sans perte pour communication bidirectionnelle",* French patent request # 1151727, March 2011 ; international patent request PCT/FR2012/050428, published WO 2012/117206 September 7th, 2012

Appendix III

This appendix represents the dictionary used for parsing the XProtocol requests by the XParser block, and converting them into their MPEG-4 BiFS and LASeR counterpart by the Semantic MPEG-4 Convertor. Their functions are written in C language and are presented in Tables 1, 2, 3 and 4

Table 1. Code extraction in C language from the libraries of the XParser and Semantic MPEG-4 blocks

XParser (requests lists) (parsing functions)	Semantic MPEG-4 BiFS Convertor (conversion functions)	Semantic MPEG-4 LASeR Convertor (conversion functions)
parseXCreateWindow(ConnectedX11Application* x11Appl, unsigned char *, unsigned char *);	X11toBIFSLib_CreateWindow(X11toBIFSLibPtr pLib, X11toBIFSLib__WindowParams* params);	X11toLASERLib_CreateWindow(X11toLASERLibPtr pLib, X11toLASERLib__WindowParams* params);
parseXChangeWindowAttributes(ConnectedX11Application* x11Appl, unsigned char*, unsigned char *);	X11toBIFSLib_ChangeWindowAttributes(X11toBIFSLibPtr pLib, X11toBIFSLib__WindowParams* params);	X11toLASERLib_ChangeWindowAttributes(X11toLASERLibPtr pLib, X11toLASERLib__WindowParams* params);
parseXDestroyWindow(ConnectedX11Application* x11Appl, unsigned char*, unsigned char *);	X11toBIFSLib_DestroyWindow(X11toBIFSLibPtr pLib, X11toBIFSLib__MapWindowParams* params);	X11toLASERLib_DestroyWindow(X11toLASERLibPtr pLib, const char* id);
parseXMapWindow(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_MapWindow(X11toBIFSLibPtr pLib, X11toBIFSLib__MapWindowParams* params);	X11toLASERLib_MapWindow(X11toLASERLibPtr pLib, X11toLASERLib__MapWindowParams* params);
parseXMapSubwindows(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_MapSubWindows(X11toBIFSLibPtr pLib, X11toBIFSLib__MapWindowParams* params);	X11toLASERLib_MapSubWindows(X11toLASERLibPtr pLib, X11toLASERLib__MapWindowParams* params);
parseXUnmapWindow(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_UnmapWindow(X11toBIFSLibPtr pLib, X11toBIFSLib__MapWindowParams* params);	X11toLASERLib_UnmapWindow(X11toLASERLibPtr pLib, X11toLASERLib__MapWindowParams* params);
parseXConfigureWindow(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_ConfigureWindow(X11toBIFSLibPtr pLib, X11toBIFSLib__ConfigureWindowParams* params);	X11toLASERLib_ConfigureWindow(X11toLASERLibPtr pLib, X11toLASERLib__ConfigureWindowParams* params);
parseXCreatePixmap(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_CreatePixmap(X11toBIFSLibPtr pLib, X11toBIFSLib__PixmapParams* params);	X11toLASERLib_CreatePixmap(X11toLASERLibPtr pLib, X11toLASERLib__PixmapParams* params);
parseXFreePixmap(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_FreePixmap(X11toBIFSLibPtr pLib, const char* id);	X11toLASERLib_FreePixmap(X11toLASERLibPtr pLib, const char* id);
parseXCreateGC(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_CreateGC(X11toBIFSLibPtr pLib, X11toBIFSLib__GCParams* params);	X11toLASERLib_CreateGC(X11toLASERLibPtr pLib, X11toLASERLib__GCParams* params);
parseXChangeGC(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_ChangeGC(X11toBIFSLibPtr pLib, X11toBIFSLib__GCParams* params);	X11toLASERLib_ChangeGC(X11toLASERLibPtr pLib, X11toLASERLib__GCParams* params);
parseXSetClipRectangles(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_SetClipRectangles(X11toBIFSLibPtr pLib, X11toBIFSLib__ClipRectanglesParams* params);	X11toLASERLib_SetClipRectangles(X11toLASERLibPtr pLib, X11toLASERLib__ClipRectanglesParams* params);
parseXFreeGC(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_FreeGC(X11toBIFSLibPtr pLib, const char* id);	X11toLASERLib_FreeGC(X11toLASERLibPtr pLib, const char* id);
parseXCopyArea(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_CopyArea(X11toBIFSLibPtr pLib, X11toBIFSLib__CopyAreaParams* params);	X11toLASERLib_CopyArea(X11toLASERLibPtr pLib, X11toLASERLib__CopyAreaParams* params);
parseXPolyLine(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_AddPolyLine(X11toBIFSLibPtr pLib, X11toBIFSLib__PolyLineParams* params);	X11toLASERLib_AddPolyLine(X11toLASERLibPtr pLib, X11toLASERLib__PolyLineParams* params);
parseXPolySegment(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_AddPolySegment(X11toBIFSLibPtr pLib, X11toBIFSLib__SegmentParams* params);	X11toLASERLib_AddPolySegment(X11toLASERLibPtr pLib, X11toLASERLib__SegmentParams* params);
parseXPolyRectangle(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_AddPolyRectangle(X11toBIFSLibPtr pLib, X11toBIFSLib__RectangleParams* params);	X11toLASERLib_AddPolyRectangle(X11toLASERLibPtr pLib, X11toLASERLib__RectangleParams* params);
parseXPolyArc(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_AddPolyArc(X11toBIFSLibPtr pLib, X11toBIFSLib__PolyArcParams* params);	X11toLASERLib_AddPolyArc(X11toLASERLibPtr pLib, X11toLASERLib__PolyArcParams* params);
parseXFillPoly(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_AddFillPoly(X11toBIFSLibPtr pLib, X11toBIFSLib__FillPolyParams* params);	X11toLASERLib_AddFillPoly(X11toLASERLibPtr pLib, X11toLASERLib__FillPolyParams* params);
parseXPolyFillRectangle(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_AddPolyFillRectangle(X11toBIFSLibPtr pLib, X11toBIFSLib__RectangleParams* params);	X11toLASERLib_AddPolyFillRectangle(X11toLASERLibPtr pLib, X11toLASERLib__RectangleParams* params);
parseXPolyFillArc(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_AddPolyArc(X11toBIFSLibPtr pLib, X11toBIFSLib__PolyArcParams* params);	X11toLASERLib_AddPolyArc(X11toLASERLibPtr pLib,X11toLASERLib__PolyArcParams* params);
parseXPutImage(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_PutImage(X11toBIFSLibPtr pLib, X11toBIFSLib__ImageParams* params);	X11toLASERLib_PutImage(X11toLASERLibPtr pLib, X11toLASERLib__ImageParams* params);
parseXPolyText8(ConnectedX11Application*, unsigned char*, unsigned char *);	X11toBIFSLib_PolyText8(X11toBIFSLibPtr pLib, X11toBIFSLib__PolyText8Params* params);	X11toLASERLib_PolyText8(X11toLASERLibPtr pLib, X11toLASERLib__PolyText8Params* params);

Table 2. Code extraction in C language from the XParser class

XProtocol request name	XParser parsing function

polyRectangle	`int X11Parser::parseXPolyRectangle(ConnectedX11Application* x11Appl,`
	`unsigned char * header, unsigned char *msg) {`
	`DEBUG(logger,"PolyRectangle");`
	`X11toBIFSLib__RectangleParams paramsBifs;`
	`X11toLASERLib__RectangleParams paramsLaser;`
	`unsigned int drawable = x11Appl->getUInt32(&(msg[0]));`
	`unsigned int GC = x11Appl->getUInt32(&(msg[4]));`
	`unsigned int noRects = (x11Appl->getUInt16(&(header[2])) - 3) / 2;`
	
	`sprintf(paramsBifs.did, "0x%x", drawable);`
	`sprintf(paramsBifs.gc, "0x%x", GC);`
	`paramsBifs.pMat = NULL;`
	
	`sprintf(paramsLaser.did, "0x%x", drawable);`
	`sprintf(paramsLaser.gc, "0x%x", GC);`
	`paramsLaser.pFillColor = NULL;`
	`paramsLaser.pStrokeColor = NULL;`
	`paramsLaser.lineWidth = -1;`
	`paramsLaser.isFillRectangle = 0;`
	
	`for (int i = 0; i < noRects; i++) {`
	
	`short x = x11Appl->getInt16(&msg[8 + 8 * i]);`
	`short y = x11Appl->getInt16(&msg[8 + 8 * i + 2]);`
	`unsigned short width = x11Appl->getInt16(&msg[8 + 8 * i + 4]);`
	`unsigned short height = x11Appl->getInt16(&msg[8 + 8 * i + 6]);`
	
	`paramsLaser.rect.x = paramsBifs.rect.x = x;`
	`paramsLaser.rect.y = paramsBifs.rect.y = y;`
	`paramsLaser.rect.width = paramsBifs.rect.width = width;`
	`paramsLaser.rect.height = paramsBifs.rect.height = height;`
	
	`TRACE(logger,"\t Rectangle no " << i);`
	`TRACE(logger,"\t\t x: " << x);`
	`TRACE(logger,"\t\t y: " << y);`
	`TRACE(logger,"\t\t width: " << width);`
	`TRACE(logger,"\t\t height: " << height);`
	
	`X11toBIFSLib_AddPolyRectangle(bifs, &paramsBifs);`
	`X11toLASERLib_AddPolyRectangle(laser, &paramsLaser);`
	`}`
	`totalrect++;`
	`fflush(fp);`
	
	`return 1;`
	`}`

polyFillRectangle	`int X11Parser::parseXPolyFillRectangle(ConnectedX11Application* x11Appl,`
	`unsigned char * header, unsigned char *msg) {`
	`X11toBIFSLib__RectangleParams paramsBifs;`
	`X11toLASERLib__RectangleParams paramsLaser;`
	`unsigned int drawable = x11Appl->getUInt32(&(msg[0]));`
	`unsigned int GC = x11Appl->getUInt32(&(msg[4]));`
	`unsigned int noRects = (x11Appl->getUInt16(&(header[2])) - 3)/2;`
	
	`sprintf(paramsBifs.did, "0x%x", drawable);`
	`sprintf(paramsBifs.gc, "0x%x", GC);`
	`sprintf(paramsLaser.did, "0x%x", drawable);`
	`sprintf(paramsLaser.gc, "0x%x", GC);`
	`paramsBifs.pMat = NULL;`
	`paramsLaser.pFillColor = NULL;`
	`paramsLaser.pStrokeColor = NULL;`
	`paramsLaser.lineWidth = -1;`
	`paramsLaser.isFillRectangle = 1;`
	`for (unsigned int i = 0; i < noRects; i++) {`
	
	`short x = x11Appl->getInt16(&msg[8 + 8 * i]);`
	`short y = x11Appl->getInt16(&msg[8 + 8 * i + 2]);`
	`unsigned short width = x11Appl->getInt16(&msg[8 + 8 * i + 4]);`
	`unsigned short height = x11Appl->getInt16(&msg[8 + 8 * i + 6]);`
	
	`paramsLaser.rect.x = paramsBifs.rect.x = x;`
	`paramsLaser.rect.y = paramsBifs.rect.y = y;`
	`paramsLaser.rect.width = paramsBifs.rect.width = width;`
	`paramsLaser.rect.height = paramsBifs.rect.height = height;`

```
                X11toBIFSLib_AddPolyFillRectangle(bifs, &paramsBifs);
                X11toLASERLib_AddPolyFillRectangle(laser, &paramsLaser);

        }
        totalFillrect++;
        fflush(fp);

        return 1;
}
```

putImage

```
int X11Parser::parseXPutImage(ConnectedX11Application* x11Appl,
                unsigned char * header, unsigned char *msg) {
        static ImageChannel* imChannel = ImageChannel::getInstance();

        X11toBIFSLib__ImageParams paramsBifs;
        X11toLASERLib__ImageParams paramsLaser;
        unsigned int format = x11Appl->getUInt8(&header[1]);
        unsigned int request_length = getRequestLength(x11Appl, header);
        unsigned int drawable = x11Appl->getUInt32(&msg[0]);
        unsigned int GC = x11Appl->getUInt32(&(msg[4]));
        unsigned int width = x11Appl->getUInt16(&msg[8]);
        unsigned int height = x11Appl->getUInt16(&msg[10]);
        unsigned int dstX = x11Appl->getUInt16(&msg[12]);
        unsigned int dstY = x11Appl->getUInt16(&msg[14]);
        unsigned int leftPad = x11Appl->getUInt8(&msg[16]);
        unsigned int depth = x11Appl->getUInt8(&msg[17]);
        char * imageInBase64ForHtml;

        int components;
        int calcsize;
        if (format != 2) {
                printf("unsupported format %d\n", format);
                return 0;
        }

        sprintf(paramsBifs.gc, "0x%x", GC);
        sprintf(paramsBifs.did, "0x%x", drawable);
        sprintf(paramsLaser.gc, "0x%x", GC);
        sprintf(paramsLaser.did, "0x%x", drawable);

        paramsLaser.h = paramsBifs.h = height;
        paramsLaser.w = paramsBifs.w = width;
        paramsLaser.x = paramsBifs.x = dstX;
        paramsLaser.y = paramsBifs.y = dstY;
        /* Format is:
         * 0   Bitmap (monochromatic, depth must be 1)
         * 1   XYPixmap
         * 2   ZPixmap */
        paramsLaser.format = paramsBifs.format = format;
        /* depth is in bits per pixel. */
        paramsLaser.dept = paramsBifs.dept = depth;

        /* We need at least one byte per pixel. */
        if (paramsBifs.dept < 8) { /* format == 0 */
                components = 1;
        } else {
                /* Add 1 for the alpha channel. */
                components = paramsBifs.dept / 8 + 1;
        }

        calcsize = paramsBifs.h * paramsBifs.w * components;
        paramsLaser.size = paramsBifs.size = (request_length - 6) * 4;
        paramsLaser.image = paramsBifs.image = (unsigned char*) msg + 20;
        paramsLaser.imageNode = paramsBifs.imageNode = NULL;
        DEBUG(logger,"PutImage on 0x" << std::hex << drawable << std::dec << "
width: " << width << " height: " << height << " (dstX,dstY)=(" << dstX << ", " <<
dstY << ")");
                totalraw += calcsize;
        fprintf(fp,"%d\t%d\n",calcsize,totalraw);
        fflush(fp);
        totalim++;
        fprintf(fp,"Img:%d\tLine:%d\tRect:%d\tFillRect:%d\n",totalim,totalline,tota
lrect,totalFillrect);
        fflush(fp);
        X11toBIFSLib_PutImage(bifs, &paramsBifs);
        X11toLASERLib_PutImage(laser, &paramsLaser);
        return 1;
}
```

polyText8

```
int X11Parser::parseXPolyText8(ConnectedX11Application* x11Appl,
                unsigned char * header, unsigned char *msg) {
        DEBUG(logger,"PolyText8");
        X11toBIFSLib__PolyText8Params paramsBifs;
        X11toLASERLib__PolyText8Params paramsLaser;
        unsigned int drawable = x11Appl->getUInt32(&msg[0]);
```

```
        unsigned int GC = x11Appl->getInt32(&(msg[4]));
        int x = x11Appl->getInt16(&msg[8]);
        int y = x11Appl->getInt16(&msg[10]);
        unsigned int itemsLength = 4 * (getRequestLength(x11Appl, header) - 4); /*
includes padding! */
        unsigned int item;
        unsigned char *cur;

        sprintf(paramsBifs.did, "0x%x", drawable);
        sprintf(paramsBifs.gc, "0x%x", GC);
        sprintf(paramsLaser.did, "0x%x", drawable);
        sprintf(paramsLaser.gc, "0x%x", GC);
        paramsLaser.x = paramsBifs.x = x;
        paramsLaser.y = paramsBifs.y = y;

        cur = &msg[12];
        paramsHtml.text = paramsLaser.text = paramsBifs.text = NULL;
        while (cur < &msg[12 + itemsLength]) {
                int len = x11Appl->getInt8(&cur[0]);
                if (len == 0)
                        break;
                if (len == 255) {
                        /* Font shift command. Ignore it for now. */
                        cur += 5;
                        continue;
                }
                int delta = x11Appl->getInt8(&cur[1]);
                char* msg = (char*) &cur[2];
                paramsBifs.text = (char*) realloc(paramsBifs.text, len + 1);
                memcpy(paramsBifs.text, msg, len);
                paramsBifs.text[len] = '\0';
                DEBUG(logger, "text is: /" << paramsBifs.text << "/");
                paramsLaser.text=(char*)realloc(paramsLaser.text,len+1);
                memcpy(paramsLaser.text, msg, len);
                paramsLaser.text[len] = '\0';
                X11toBIFSLib_PolyText8(bifs, &paramsBifs);
                X11toLASERLib_PolyText8(laser, &paramsLaser);
                X11toHTMLLib_PolyText8(html, &paramsHtml);
                cur += len + 2;
        }
        if (paramsBifs.text)
                free(paramsBifs.text);
        if (paramsLaser.text)
                        free(paramsLaser.text);
        return 1;
}
```

polySegment

```
int X11Parser::parseXPolySegment(ConnectedX11Application* x11Appl,
                unsigned char * header, unsigned char *msg) {
        DEBUG(logger,"PolySegment");
        X11toBIFSLib__SegmentParams paramsBifs;
        X11toLASERLib__SegmentParams paramsLaser;
        unsigned int drawable = x11Appl->getUInt32(&(msg[0]));
        unsigned int GC = x11Appl->getUInt32(&(msg[4]));
        unsigned int noSegments = (x11Appl->getUInt16(&(header[2])) - 3) / 2;

        sprintf(paramsBifs.did, "0x%x", drawable);
        sprintf(paramsBifs.gc, "0x%x", GC);

        sprintf(paramsLaser.did, "0x%x", drawable);
        sprintf(paramsLaser.gc, "0x%x", GC);
        paramsBifs.x = (int*) malloc(2 * sizeof(int));
        paramsBifs.y = (int*) malloc(2 * sizeof(int));

        paramsLaser.x = (int*) malloc(2 * sizeof(int));
        paramsLaser.y = (int*) malloc(2 * sizeof(int));
        for (int i = 0; i < noSegments; i++) {
                int x1 = x11Appl->getInt16(&msg[8 + 8 * i]);
                int y1 = x11Appl->getInt16(&msg[8 + 8 * i + 2]);
                int x2 = x11Appl->getInt16(&msg[8 + 8 * i + 4]);
                int y2 = x11Appl->getInt16(&msg[8 + 8 * i + 6]);

                paramsLaser.x[0] = paramsBifs.x[0] = x1;
                paramsLaser.y[0] = paramsBifs.y[0] = y1;
                paramsLaser.x[1] = paramsBifs.x[1] = x2;
                paramsLaser.y[1] = paramsBifs.y[1] = y2;

                X11toBIFSLib_AddPolySegment(bifs, &paramsBifs);
                X11toLASERLib_AddPolySegment(laser, &paramsLaser);
        }
        totalline++;
        fflush(fp);
        free(paramsBifs.x);
        free(paramsBifs.y);
```

```
        free(paramsLaser.x);
        free(paramsLaser.y);
        return 1;
}
```

copyArea

```
int X11Parser::parseXCopyArea(ConnectedX11Application* x11Appl,
                unsigned char * header, unsigned char *msg) {
        X11toBIFSLib__CopyAreaParams paramsBifs;
        X11toLASERLib__CopyAreaParams paramsLaser;

        unsigned int srcDrawable = x11Appl->getUInt32(&(msg[0]));
        unsigned int dstDrawable = x11Appl->getUInt32(&(msg[4]));
        unsigned int GC = x11Appl->getUInt32(&(msg[8]));
        int srcX = x11Appl->getInt16(&(msg[12]));
        int srcY = x11Appl->getInt16(&(msg[14]));
        int dstX = x11Appl->getInt16(&(msg[16]));
        int dstY = x11Appl->getInt16(&(msg[18]));
        unsigned int width = x11Appl->getUInt16(&(msg[20]));
        unsigned int height = x11Appl->getUInt16(&(msg[22]));

        sprintf(paramsBifs.dst_did, "0x%x", dstDrawable);
        sprintf(paramsBifs.src_did, "0x%x", srcDrawable);
        sprintf(paramsBifs.gc, "0x%x", GC);
        sprintf(paramsLaser.dst_did, "0x%x", dstDrawable);
        sprintf(paramsLaser.src_did, "0x%x", srcDrawable);
        sprintf(paramsLaser.gc, "0x%x", GC);
        paramsHtml.src_x =paramsLaser.src_x = paramsBifs.src_x = srcX;
        paramsHtml.src_y =paramsLaser.src_y = paramsBifs.src_y = srcY;
        paramsHtml.dst_y =paramsLaser.dst_y = paramsBifs.dst_y = dstY;
        paramsHtml.dst_x =paramsLaser.dst_x = paramsBifs.dst_x = dstX;
        paramsLaser.w = paramsBifs.w = width;
        paramsLaser.h = paramsBifs.h = height;

        X11toBIFSLib_CopyArea(bifs, &paramsBifs);
        X11toLASERLib_CopyArea(laser, &paramsLaser);

        return 0;
}
```

createGC

```
int X11Parser::parseXCreateGC(ConnectedX11Application* x11Appl,
                unsigned char * header, unsigned char *msg) {
        DEBUG(logger,"CreateGC");
        X11toBIFSLib__GCParams paramsBifs;
        X11toLASERLib__GCParams paramsLaser;

        unsigned int cid = x11Appl->getUInt32(msg);
        unsigned int drawable = x11Appl->getUInt32(&(msg[4]));
        unsigned int bitmask = x11Appl->getUInt32(&(msg[8]));

        memset(&paramsBifs, 0, sizeof(paramsBifs));
        sprintf(paramsBifs.did, "0x%x", drawable);
        sprintf(paramsBifs.cid, "0x%x", cid);
        paramsBifs.isFore = 1;
        paramsBifs.background = 1;
        paramsBifs.isLineW = 0;
        paramsBifs.line_width = 0;
        paramsBifs.isCapS = 1;
        paramsBifs.cap_style = 1;
        paramsBifs.isArcM = 1;
        paramsBifs.arc_mode = 4;
        paramsBifs.isDash = 1;
        paramsBifs.dashes = 4;
        parseGCBitmaskBifs(bitmask, &msg[12], &paramsBifs);
        X11toBIFSLib_CreateGC(bifs, &paramsBifs);

        memset(&paramsLaser, 0, sizeof(paramsLaser));
        sprintf(paramsLaser.did, "0x%x", drawable);
        sprintf(paramsLaser.cid, "0x%x", cid);
        // Default non-zero values.
        paramsLaser.isFore = 1;
        paramsLaser.background = 1;
        paramsLaser.isLineW = 0;
        paramsLaser.line_width = 0;
        paramsLaser.isCapS = 1;
        paramsLaser.cap_style = 1;
        paramsLaser.isArcM = 1;
        paramsLaser.arc_mode = 4;
        paramsLaser.isDash = 1;
        paramsLaser.dashes = 4;
        parseGCBitmaskLaser(bitmask, &msg[12], &paramsLaser);
        X11toLASERLib_CreateGC(laser, &paramsLaser);

        return 1;
}
```

Table 3. Code extraction in C language from the BiFS conversion classes

XParser parsing function	Conversion to BiFS function
parseXPolyRectangle	

```
int X11toBIFSLib_AddPolyRectangle(X11toBIFSLibPtr pLib,
                  X11toBIFSLib__RectangleParams* params) {
         X11toBIFSLib *lib = (X11toBIFSLib*)pLib;
         GF_SceneGraph *pScene = (GF_SceneGraph*)lib->pScene;
         M_Transform2D *pTr;
         M_Shape *pShp;
         GF_VRMLParent *pFirst;
         GF_VRMLParent *pNode;
         M_Material2D *pMat;
         M_Material2D *pMat1;
         M_Appearance *pApp;
         M_Rectangle *pRec;
         GF_CommandField *inf;
         GF_Command* InsertCmd;
         int ID;
         char name[15];
         pFirst = (GF_VRMLParent*)FindDrawable(lib, "0x00000000");
         if ( ! (pNode = (GF_VRMLParent*)FindDrawable(lib, params-
>did)))
                  pNode = (GF_VRMLParent*)pFirst;
         pTr = (M_Transform2D*)gf_node_new(pScene,
TAG_MPEG4_Transform2D);
         DEBUG_PRINTF("Transform2D: %d %p\n",
gf_node_get_id((GF_Node*)pTr), pTr);
         gf_node_list_add_child(&(pNode->children), (GF_Node*)pTr);
         gf_node_register((GF_Node*)pTr, (GF_Node*)pNode);
         pTr->translation.x = (Fixed)params->rect.x + params-
>rect.width/2.f;
         pTr->translation.y = - ((Fixed)params->rect.y + params-
>rect.height/2.f);

         DEBUG_PRINTF("AddPolyRectangle: %dx%d @ (%d,%d), translation:
(%f,%f)\n",
                 params->rect.width, params->rect.height, params->rect.x,
                 params->rect.y, pTr->translation.x, pTr->translation.y);

         pShp = (M_Shape*)gf_node_new(pScene, TAG_MPEG4_Shape);
         DEBUG_PRINTF("Shape: %d %p\n", gf_node_get_id((GF_Node*)pShp),
pShp);
         gf_node_list_add_child(&(pTr->children), (GF_Node*)pShp);
         gf_node_register((GF_Node*)pShp, (GF_Node*)pTr);

         if (params->pMat)
                  pMat = params->pMat;
         else
                  pMat = (M_Material2D*)FindGC(lib, params->gc, 0);

         if (pMat) {
                  pApp = (M_Appearance*)gf_node_new(pScene,
TAG_MPEG4_Appearance);
                  DEBUG_PRINTF("Appearance: %d %p\n",
gf_node_get_id((GF_Node*)pApp), pApp);
                  pShp->appearance = (GF_Node*)pApp;
                  gf_node_register( (GF_Node*)pApp, (GF_Node*)pShp );

                  pMat1 = (M_Material2D*)gf_node_new(pScene,
TAG_MPEG4_Material2D);
                  DEBUG_PRINTF("Material2D: %d %p\n",
gf_node_get_id((GF_Node*)pMat1), pMat1);
                  pMat1->filled=0;
                  pMat1->lineProps = pMat->lineProps;
                  gf_node_register((GF_Node*)pMat->lineProps,
(GF_Node*)pMat1);
                  DEBUG_PRINTF("line_width == %f\n",
                                    ((M_XLineProperties*)(pMat1-
>lineProps))->width);
                  pApp->material = (GF_Node*)pMat1;
                  gf_node_register((GF_Node*)pMat1, (GF_Node*)pApp );
         }
         pRec = (M_Rectangle*)gf_node_new(pScene, TAG_MPEG4_Rectangle);
         gf_node_register((GF_Node*)pRec, (GF_Node*)pShp);
         pShp->geometry = (GF_Node*)pRec;
         pRec->size.x = (Fixed)params->rect.width;
         pRec->size.y = (Fixed)params->rect.height;
```

	`if ( !IsNodeInGraph((GF_Node*)pNode) )`
	`return 0;`
	`InsertCmd = gf_sg_command_new(pScene, GF_SG_NODE_INSERT);`
	`InsertCmd->node = (GF_Node *)pNode;`
	`gf_node_register((GF_Node*)pNode, NULL);`
	`inf = gf_sg_command_field_new(InsertCmd);`
	`inf->pos = -1;`
	`inf->new_node = (GF_Node *)pTr;`
	`gf_node_register((GF_Node*)inf->new_node, NULL);`
	`ID = gf_sg_get_next_available_node_id(pScene);`
	`sprintf(name, "PREC%d", ID);`
	`gf_node_set_id( (GF_Node*)inf->new_node, ID, name);`
	`DEBUG_PRINTF("Rectangle: %d %p\n",`
	`gf_node_get_id((GF_Node*)pRec), pRec);`
	`inf->field_ptr = &inf->new_node;`
	`inf->fieldType = GF_SG_VRML_SFNODE;`
	`gf_list_add(lib->cmdList, InsertCmd);`
	`X11toBIFSLib_SaveMP4(lib);`
	`return 0;`
	`}`

parseXPolyFillRectangle	`int X11toBIFSLib_AddPolyFillRectangle(X11toBIFSLibPtr pLib,`
	`X11toBIFSLib__RectangleParams* params) {`
	`X11toBIFSLib *lib = (X11toBIFSLib*)pLib;`
	`GCMapPtr pMap;`
	`M_Material2D *pMat;`
	`GF_VRMLParent *pFirst;`
	`GF_VRMLParent *pNode;`
	`pFirst = (GF_VRMLParent*)FindDrawable(lib, "0x00000000");`
	`if (!(pNode = (GF_VRMLParent*)FindDrawable(lib, params->did)))`
	`pNode = (GF_VRMLParent*)pFirst;`
	`if (params->pMat) {`
	`pMap = FindGCMapByMaterial(lib, params->pMat);`
	`assert(pMap);`
	`assert(pMap->pMaterial == params->pMat);`
	`pMat = params->pMat;`
	`} else {`
	`pMap = FindGCMap(lib, params->gc, 0);`
	`pMat = pMap->pMaterial;`
	`}`
	`DEBUG_PRINTF("pMat (GC with id %s): %d %p\n", params->gc,`
	`gf_node_get_id((GF_Node*)pMat),`
	`pMat);`
	`if (!pMap->noRects) {`
	`// Standard AddPolyFillRectangle, with no clip mask.`
	`putRectangle(lib, pNode, pMat, params);`
	`} else {`
	`int i;`
	`for (i = 0; i < pMap->noRects; ++i) {`
	`X11toBIFSLib__Rectangle* paramsRect =`
	`&(params->rect);`
	`X11toBIFSLib__Rectangle* maskRect = &(pMap-`
	`>rects[i]);`
	`X11toBIFSLib__RectangleParams newParams;`
	`X11toBIFSLib__Rectangle* newRect =`
	`&newParams.rect;`
	
	`int endParamsRect = paramsRect->x +`
	`paramsRect->width;`
	`int endMaskRect = maskRect->x + maskRect-`
	`>width;`
	`int endNewRect = MIN(endParamsRect,`
	`endMaskRect);`
	`strcpy(newParams.did, params->did);`
	`strcpy(newParams.gc, params->gc);`
	`newRect->x = MAX(paramsRect->x, maskRect->x);`
	`newRect->width = endNewRect - newRect->x;`
	`DEBUG_PRINTF("[X] Params: %d+%d; mask: %d+%d;`
	`result: %d+%d\n",`
	`paramsRect->x, paramsRect-`
	`>width,`
	`maskRect->x, maskRect-`
	`>width,`
	`newRect->x, newRect->width);`
	`endParamsRect = paramsRect->y + paramsRect-`
	`>height;`
	`endMaskRect = maskRect->y + maskRect->height;`
	`endNewRect = MIN(endParamsRect, endMaskRect);`
	`newRect->y = MAX(paramsRect->y, maskRect->y);`
	`newRect->height = endNewRect - newRect->y;`
	`DEBUG_PRINTF("[Y] Params: %d+%d; mask: %d+%d;`
	`result: %d+%d\n",`
	
	`paramsRect->y, paramsRect->height,`

	`maskRect->y, maskRect->height,`
	
	`newRect->y, newRect->height);`
	`if (newRect->width > 0 && newRect->height > 0)`
	`putRectangle(lib, pNode, pMat, &newParams);`
	`}`
	`}`
	`X11toBIFSLib_SaveMP4(lib);`
	`return 0;`
	`}`

parseXPutImage	`int X11toBIFSLib_PutImage(X11toBIFSLibPtr pLib,`
	`X11toBIFSLib__ImageParams* params) {`
	`X11toBIFSLib *lib = (X11toBIFSLib*)pLib;`
	`GF_SceneGraph *pScene = (GF_SceneGraph*)lib->pScene;`
	`M_Transform2D *pTr;`
	`M_Shape *pShp;`
	`M_Appearance *pApp;`
	`GF_VRMLParent *pFirstNode;`
	`GF_VRMLParent *pNode;`
	`M_Rectangle *pRectangle;`
	`int ID;`
	`char name[16];`
	`int movieIndex;`
	`float translationX;`
	`float translationY;`
	`#ifdef USE PRUNER`
	`NodeIDMapPtr pMap;`
	`#endif`
	
	`translationX = (Fixed)params->x + params->w/2.f;`
	`translationY = - ((Fixed)params->y + params->h/2.f);`
	
	`DEBUG_PRINTF("PutImage: %dx%dx%d @ (%d,%d), translation: (%f,%f)\n",`
	`params->w, params->h, params->dept, params->x, params->y,`
	`translationX, translationY);`
	
	`if (params->dept != 24) {`
	`DEBUG_PRINTF("Unsupported image depth %d, ignoring PutImage.\n",`
	`params->dept);`
	`return 0;`
	`}`
	`pTr = (M_Transform2D*)gf_node_new(pScene, TAG_MPEG4_Transform2D);`
	`ID = gf_sg_get_next_available_node_id(pScene);`
	`sprintf(name, "PI%d", ID);`
	`gf_node_set_id((GF_Node*)pTr, ID, name);`
	`DEBUG_PRINTF("Transform2D: %d %p\n", gf_node_get_id((GF_Node*)pTr), pTr);`
	`gf_node_list_add_child(&(pNode->children), (GF_Node*)pTr);`
	`gf_node_register((GF_Node*)pTr, (GF_Node*)pNode);`
	`pTr->translation.x = translationX;`
	`pTr->translation.y = translationY;`
	`if (!textLinesProps && (IsNodeInGraph((GF_Node*)pNode))) {`
	`TextLinesHash_Insert(lib->textLinesHash,`
	`params->x, params->y, params->w, params->h, pTr);`
	`}`
	`pShp = (M_Shape*)gf_node_new(pScene, TAG_MPEG4_Shape);`
	`DEBUG_PRINTF("Shape: %d %p\n", gf_node_get_id((GF_Node*)pShp), pShp);`
	`gf_node_list_add_child(&(pTr->children), (GF_Node*)pShp);`
	`gf_node_register((GF_Node*)pShp, (GF_Node*)pTr);`
	
	`pApp = (M_Appearance*)gf_node_new(pScene, TAG_MPEG4_Appearance);`
	`DEBUG_PRINTF("Appearance: %d %p\n", gf_node_get_id((GF_Node*)pApp), pApp);`
	`pShp->appearance = (GF_Node*)pApp;`
	`gf_node_register((GF_Node*)pApp, (GF_Node*)pShp);`
	
	`addEmptyMaterial(pLib, pApp);`
	
	`int err;`
	`if (!IsNodeInGraph((GF_Node*)pNode)) {`
	`// Add PixelTexture`
	`assert(params->image);`
	`err = putPixelTexture(lib, params, pApp,0);`
	`} else if (!params->image) {`
	`abort();`

```
                assert(params->imageNode);

                err = putUsePixelTexture(lib, params, pApp,
                                (M_PixelTexture*)params->imageNode);
        } else {
                // CopyArea from out-of-scene to scene.
                unsigned char* image = malloc(params->w * params->h *
3);
                assert(!params->imageNode);
                copyPixmapX11toRGB(params->image, image, params->w,
                                params->h,get_X11_bit_padding(params-
>dept) / 8, 3);
                M_PixelTexture* pPixText =
                                (M_PixelTexture*)ImageHash_Find(lib-
>imageHash, params->w,
                                                        params->h, 3,
image);
                if (pPixText) {

                        err = putUsePixelTexture(lib, params, pApp,
pPixText);
                        free(image);
                } else {

                        err = putPixelTexture(lib, params, pApp,1);
                }
        }
#if 0
                err = encodeImage(params);
                if (err)
                        return err;
                err = putPNGPixelTexture(lib, params, pApp);
        }
#endif

#if 0
                char* image_id = ImageHash_Find(lib->imageHash,
params->w, params->h, 3,
                                params->image);
                if (strcmp(image_id, "") != 0) {
                        // Already in hash.
                        err = putImageTexture(lib, params, pApp,
image_id);
#endif
        if (err)
                return err;
        pRectangle = (M_Rectangle*)gf_node_new(pScene,
TAG_MPEG4_Rectangle);
        ID = gf_sg_get_next_available_node_id(pScene);
        sprintf(name, "PIR%d", ID);
        gf_node_set_id((GF_Node*)pRectangle, ID, name);
        DEBUG_PRINTF("Rectangle: %d %p\n",
gf_node_get_id((GF_Node*)pRectangle), pRectangle);
        gf_node_register((GF_Node*)pRectangle, (GF_Node*)pShp);
        pShp->geometry = (GF_Node*)pRectangle;
        pRectangle->size.x = params->w;
        pRectangle->size.y = params->h;

        if (addInsertCommand(lib, (GF_Node*)pNode, (GF_Node*)pTr, -1))
{
//#ifdef use PRUNER
                if (IsNodeInGraph((GF_Node*)pNode)) {
                        pMap = FindWindow(lib, params->did);
                        assert(pMap);
                        handleCoverageHandler(lib, params->x, params-
>y, params->w, params->h, pMap->coverage_handler, pTr);
                }

                lib->forceSave = 1;
        X11toBIFSLib_SaveMP4(lib);
        }
        return 0;
}
```

parseXPolyText8

```
int X11toBIFSLib_PolyText8(X11toBIFSLibPtr
pLib,X11toBIFSLib__PolyText8Params* params) {
        X11toBIFSLib *lib = (X11toBIFSLib*)pLib;
        GF_SceneGraph *pScene = (GF_SceneGraph*)lib->pScene;
        M_Transform2D *pTr;
        M_Shape *pShp;
        M_Appearance *pApp;
        GF_VRMLParent *pFirstNode;
        GF_VRMLParent *pNode;
        M_Text *pText;
        M_FontStyle *pStyle;
```

	`M_Material2D *pMat;`
	`int ID;`
	`char name[15];`
	`unsigned int font_size = 20;`
	`pFirstNode = (GF_VRMLParent*)FindDrawable(lib, "0x00000000");`
	
	`if ( ! (pNode = (GF_VRMLParent*)FindDrawable(lib, params->did)))`
	`pNode = (GF_VRMLParent*)pFirstNode;`
	
	`pTr = (M_Transform2D*)gf_node_new(pScene, TAG_MPEG4_Transform2D);`
	`ID = gf_sg_get_next_available_node_id(pScene);`
	`sprintf(name, "PTX%d", ID);`
	`gf_node_set_id((GF_Node*)pTr, ID, name);`
	`DEBUG_PRINTF("Transform2D: %d %p\n", gf_node_get_id((GF_Node*)pTr), pTr);`
	`gf_node_list_add_child(&(pNode->children), (GF_Node*)pTr);`
	`gf_node_register((GF_Node*)pTr, (GF_Node*)pNode);`
	`pTr->translation.x = (Fixed)params->x;`
	`pTr->translation.y = - (Fixed)params->y;`
	
	`pShp = (M_Shape*)gf_node_new(pScene, TAG_MPEG4_Shape);`
	`gf_node_list_add_child(&(pTr->children), (GF_Node*)pShp);`
	`gf_node_register((GF_Node*)pShp, (GF_Node*)pTr);`
	
	`pMat = (M_Material2D*)FindGC(lib, params->gc, 0);`
	`if (pMat) {`
	`pApp = (M_Appearance*)gf_node_new(pScene, TAG_MPEG4_Appearance);`
	`DEBUG_PRINTF("Appearance: %d %p\n", gf_node_get_id((GF_Node*)pApp), pApp);`
	`pShp->appearance = (GF_Node*)pApp;`
	`gf_node_register( (GF_Node*)pApp, (GF_Node*)pShp );`
	
	`pApp->material = (GF_Node*)pMat;`
	`gf_node_register( (GF_Node*)pMat, (GF_Node*)pApp );`
	`}`
	
	`pText = (M_Text*)gf_node_new(pScene, TAG_MPEG4_Text);`
	`DEBUG_PRINTF("Text: %d %p\n", gf_node_get_id((GF_Node*)pText), pText);`
	`gf_node_register((GF_Node*)pText, (GF_Node*)pShp);`
	`pShp->geometry = (GF_Node*)pText;`
	`pText->string.count = 1;`
	`pText->string.vals = malloc(pText->string.count*sizeof(SFString));`
	`pText->string.vals[0] = strdup(params->text);`
	
	`pStyle = (M_FontStyle*)gf_node_new(pScene, TAG_MPEG4_FontStyle);`
	`DEBUG_PRINTF("FontStyle: %d %p\n", gf_node_get_id((GF_Node*)pStyle), pStyle);`
	`pText->fontStyle = (GF_Node*)pStyle;`
	`gf_node_register((GF_Node*)pStyle, (GF_Node*)pText);`
	`pStyle->size = font_size;`
	`pStyle->leftToRight = 1;`
	`pStyle->horizontal = 1;`
	`pStyle->family.count = 1;`
	`pStyle->family.vals = malloc(pStyle->family.count*sizeof(MFString));`
	`pStyle->family.vals[0] = strdup("courier");`
	
	`if (addInsertCommand(lib, (GF_Node*)pNode, (GF_Node*)pTr, -1))`
	`X11toBIFSLib_SaveMP4(lib);`
	`return 0;`
	`}`

parseXPolySegment	`int X11toBIFSLib_AddPolySegment(X11toBIFSLibPtr pLib, X11toBIFSLib__SegmentParams* params) {`
	`X11toBIFSLib *lib = (X11toBIFSLib*)pLib;`
	`GF_SceneGraph *pScene = (GF_SceneGraph*)lib->pScene;`
	`M_Transform2D *pTr;`
	`M_Shape *pShp;`
	`M_IndexedLineSet2D *pIfs;`
	`M_Coordinate2D *pCoord;`
	`GF_VRMLParent *pFirst;`
	`GF_VRMLParent *pNode;`
	`M_Appearance *pApp;`
	`M_Material2D *pMat;`
	`GF_CommandField *inf;`
	`GF_Command* InsertCmd;`
	`pFirst = (GF_VRMLParent*)FindDrawable(lib, "0x00000000");`
	`if ( ! (pNode = (GF_VRMLParent*)FindDrawable(lib, params->did)))`

```
                    pNode = (GF_VRMLParent*)pFirst;
          pTr = (M_Transform2D*)gf_node_new(pScene,
TAG_MPEG4_Transform2D);
          DEBUG_PRINTF("Transform2D: %d %p\n",
gf_node_get_id((GF_Node*)pTr), pTr);
          gf_node_list_add_child(&(pNode->children), (GF_Node*)pTr);
          gf_node_register((GF_Node*)pTr, (GF_Node*)pNode);

          pShp = (M_Shape*)gf_node_new(pScene, TAG_MPEG4_Shape);
          DEBUG_PRINTF("Shape: %d %p\n", gf_node_get_id((GF_Node*)pShp),
pShp);
          gf_node_list_add_child(&(pTr->children), (GF_Node*)pShp);
          gf_node_register((GF_Node*)pShp, (GF_Node*)pTr);
          pMat = (M_Material2D*)FindGC(lib, params->gc, 0);
          if (pMat) {
                    pApp = (M_Appearance*)gf_node_new(pScene,
TAG_MPEG4_Appearance);
                    DEBUG_PRINTF("Appearance: %d %p\n",
gf_node_get_id((GF_Node*)pApp), pApp);
                    pShp->appearance = (GF_Node*)pApp;
                    gf_node_register( (GF_Node*)pApp, (GF_Node*)pShp );
                    pApp->material = (GF_Node*)pMat;
                    gf_node_register( (GF_Node*)pMat, (GF_Node*)pApp );
          }
          pIfs = (M_IndexedLineSet2D*)gf_node_new(pScene,
TAG_MPEG4_IndexedLineSet2D);
          DEBUG_PRINTF("IndexedLineSet2D: %d %p\n",
gf_node_get_id((GF_Node*)pIfs), pIfs);
          gf_node_register((GF_Node*)pIfs, (GF_Node*)pShp);
          pShp->geometry = (GF_Node*)pIfs;
          pCoord = (M_Coordinate2D*)gf_node_new(pScene,
TAG_MPEG4_Coordinate2D);
          DEBUG_PRINTF("Coordinate2D: %d %p\n",
gf_node_get_id((GF_Node*)pCoord), pCoord);
          gf_node_register((GF_Node*)pCoord, (GF_Node*)pIfs);
          pIfs->coord = (GF_Node*)pCoord;
          pCoord->point.count = 2;
          pCoord->point.vals = (SFVec2f*)malloc(pCoord->point.count *
sizeof(SFVec2f));
          pCoord->point.vals[0].x = (Fixed)params->x[0];
          pCoord->point.vals[0].y = -(Fixed)params->y[0];
          pCoord->point.vals[1].x = (Fixed)params->x[1];
          pCoord->point.vals[1].y = -(Fixed)params->y[1];
          DEBUG_PRINTF("AddPolySegment: (%d, %d) -> (%d, %d) ===> (%f,
%f) -> (%f, %f)\n",
                              params->x[0], params->y[0], params->x[1],
params->y[1],
                              pCoord->point.vals[0].x, pCoord-
>point.vals[0].y,
                              pCoord->point.vals[1].x, pCoord-
>point.vals[1].y);
          if ( !IsNodeInGraph((GF_Node*)pNode) )
                    return 0;
          InsertCmd = gf_sg_command_new(pScene, GF_SG_NODE_INSERT);
          InsertCmd->node = (GF_Node *)pNode;
          gf_node_register((GF_Node*)pNode, NULL);
          inf = gf_sg_command_field_new(InsertCmd);
          inf->pos = -1;
          inf->new_node = (GF_Node *)pTr;
          gf_node_register((GF_Node*)pTr, NULL);
          inf->field_ptr = &inf->new_node;
          inf->fieldType = GF_SG_VRML_SFNODE;
          gf_list_add(lib->cmdList, InsertCmd);
          X11toBIFSLib_SaveMP4(lib);
          return 0;
}
```

parseXCopyArea

```
int X11toBIFSLib_CopyArea(X11toBIFSLibPtr pLib,
          X11toBIFSLib__CopyAreaParams* params) {
          X11toBIFSLib__RectangleParams params_Rectangle;
          X11toBIFSLib *lib = (X11toBIFSLib*)pLib;
          GF_VRMLParent *pNode_src;
          GF_VRMLParent *pNode_dst;
          GF_VRMLParent *pNode_cur;
          M_Transform2D *pTr;
          M_Shape *pShp;
          GF_VRMLParent *pFirst;
          GF_VRMLParent *node;
          M_Appearance *pApp;
          M_Rectangle *pRec;
          M_Material2D *pMat;
          GF_Node *nodeTag;
          Fixed x = 0.f;
          Fixed y = 0.f;
```

```
        int i;
        int count;
        pFirst = (GF_VRMLParent*)FindDrawable(lib, "0x00000000");
        if ( ! (pNode_src = (GF_VRMLParent*)FindDrawable(lib, params-
>src_did))) {
                DEBUG_PRINTF("Src Draw not found : %s\n", params-
>src_did);
                pNode_src = (GF_VRMLParent*)pFirst;
        }
        if ( ! (pNode_dst = (GF_VRMLParent*)FindDrawable(lib, params-
>dst_did))) {
                DEBUG_PRINTF("Dst Draw not found : %s\n", params-
>dst_did);
                pNode_dst = (GF_VRMLParent*)pFirst;
        }
        pNode_cur = pNode_src;
        count = gf_node_list_get_count(pNode_cur->children);
        DEBUG_PRINTF("CopyArea: copy (%d, %d) %dx%d to (%d, %d),
drawables %p -> %p, "
                                        "GC %s\n",
                        params->src_x, params->src_y, params->w,
params->h, params->dst_x, params->dst_y,
                        pNode_src, pNode_dst, params->gc);
        for (i = 0; i < count; i++) {
                node =
(GF_VRMLParent*)gf_node_list_get_child(pNode_cur->children, i);
        if ( node ){
        if (gf_node_get_tag((GF_Node*)node) == TAG_MPEG4_Transform2D) {
        pTr = (M_Transform2D*)node;
                x = pTr->translation.x;
                y = pTr->translation.y;
                DEBUG_PRINTF("CopyArea: found Transform2D(%f, %f)\n",
x, y);
                if ((x < (params->src_x + params->w)) &&
                                (y > - (params->src_y + params->h))
&&
                                (x >= params->src_x) &&
                                (y <= - params->src_y)) {
                        nodeTag =
(GF_Node*)gf_node_list_get_child(node->children, 0);
                if (nodeTag && gf_node_get_tag(nodeTag)     ==
TAG_MPEG4_Transform2D)
                {
                pNode_cur = (GF_VRMLParent*)node;
                i = 0;
                count = gf_node_list_get_count(pNode_cur->children);
                node =
(GF_VRMLParent*)gf_node_list_get_child(pNode_cur->children, 0);
                nodeTag = (GF_Node*)gf_node_list_get_child(node-
>children,0);
                }
                if (nodeTag && gf_node_get_tag(nodeTag) ==
TAG_MPEG4_Shape) {
                pShp = (M_Shape*)nodeTag;
                nodeTag = pShp->geometry;
                if (nodeTag && gf_node_get_tag(nodeTag)     ==
TAG_MPEG4_IndexedLineSet2D)
                {
                } else {//fill poly
                if (nodeTag && gf_node_get_tag(nodeTag)==
TAG_MPEG4_IndexedFaceSet2D) {
                } else {  // PutImage.
                        if (nodeTag && gf_node_get_tag(nodeTag)     ==
TAG_MPEG4_Rectangle) {
                        pApp = (M_Appearance*)pShp->appearance;
                        if (pApp) {
                        nodeTag = pApp->texture;
                        if (nodeTag) {
                        if (gf_node_get_tag(nodeTag)==
TAG_MPEG4_PixelTexture) {
                        // PixelTexture means no URL but "old style"
raw pixmap.
                        X11toBIFSLib__ImageParams params_image;
                        unsigned int padding;
                        unsigned int image_size;
                        M_PixelTexture *mPixTex =
(M_PixelTexture*)nodeTag;
                        Fixed old_y = -y;
                        DEBUG_PRINTF("FOOd %f %f %d\n", old_y, y,
mPixTex->image.height);
                        strcpy(params_image.did,params->dst_did);
                        strcpy(params_image.gc, params->gc);
                        params_image.h = mPixTex->image.height;
                        params_image.w = mPixTex->image.width;
```

```
                                        params_image.x = params->dst_x + x - mPixTex-
>image.width/2.f - params->src_x;
                                        params_image.y = params->dst_y + y - mPixTex-
>image.height/2.f - params->src_y;
                                        params_image.y = params->dst_y + old_y -
mPixTex->image.height/2.f - params->src_y;
                                        DEBUG_PRINTF("CopyArea: [PixelTexture] %dx%d,
translations were (%f, %f -> %f) -> placement is (%d, %d)\n",
                                        params_image.w, params_image.h, x, y, old_y,
params_image.x, params_image.y);
                                        params_image.dept = mPixTex-
>image.numComponents * 8;
                                        padding =
get_X11_bit_padding(params_image.dept) / 8;
                                        image_size = mPixTex->image.height * mPixTex-
>image.width;
                                        params_image.size = image_size * padding;
                                        params_image.image =
malloc(params_image.size);
                                        params_image.imageNode = NULL;
                                        copyPixmapRGBtoX11(mPixTex->image.pixels,
params_image.image, mPixTex->image.width,
                                        mPixTex->image.height, padding, mPixTex-
>image.numComponents);
                                        DEBUG_PRINTF("PutImage\n");
                                        X11toBIFSLib_PutImage(pLib, &params_image);
                                        free(params_image.image);
                                        } else if (gf_node_get_tag(nodeTag) ==
TAG_MPEG4_ImageTexture) {
                                        abort();
                                        M_ImageTexture *mImageText =
(M_ImageTexture*)nodeTag;
                                        X11toBIFSLib__ImageParams params_image;
                                        M_Appearance *pApp =
(M_Appearance*)gf_node_get_parent((GF_Node*)mImageText, 0);
                                        M_Shape *pShape =
(M_Shape*)gf_node_get_parent((GF_Node*)pApp, 0);
                                        M_Rectangle *pRectangle =
(M_Rectangle*)pShape->geometry;
                                        Fixed old_y = -y;
                                        strcpy(params_image.did, params->dst_did);
                                        strcpy(params_image.gc, params->gc);
                                        params_image.h = pRectangle->size.y;
                                        params_image.w = pRectangle->size.x;
                                        params_image.x = params->dst_x + x -
pRectangle->size.x/2.f -
                                        params->src_x;
                                        params_image.y = params->dst_y + old_y -
pRectangle->size.y/2.f -
                                        params->src_y;
                                        DEBUG_PRINTF("CopyArea: [ImageTexture] %dx%d,
translations were (%f, %f -> %f) -> placement is (%d, %d)\n",
                                        params_image.w, params_image.h, x, y, old_y,
params_image.x, params_image.y);
                                        params_image.imageNode = NULL;
                                        params_image.image = NULL;
                                        DEBUG_PRINTF("PutImage\n");

                                        X11toBIFSLib_PutImage(pLib, &params_image);
                                        }
                                }
                        }
                        return 0;
}
```

parseXCreateGC

```
int X11toBIFSLib_CreateGC(X11toBIFSLibPtr pLib, X11toBIFSLib__GCParams*
params) {
        X11toBIFSLib *lib = (X11toBIFSLib*)pLib;
        GF_SceneGraph *pScene = (GF_SceneGraph*)lib->pScene;
        GCMapPtr pMap;
        M_Material2D *pMat;
        M_XLineProperties *pLine;
        pMap = (GCMapPtr)malloc(sizeof(GCMap));
        strcpy(pMap->id, params->cid);
        strcpy(pMap->did, params->did);
        pMap->noRects = 0;
        pMap->rects = NULL;
        gf_list_add(lib->lstGC, pMap);
        pMat = (M_Material2D*)gf_node_new(pScene,
TAG_MPEG4_Material2D);
        gf_node_register((GF_Node*)pMat, NULL);
        gf_node_set_id((GF_Node*)pMat,

gf_sg_get_next_available_node_id(pScene), params->cid);
        DEBUG_PRINTF("Material2D: %d %p for GC %s in drawable %s\n",
```

	gf_node_get_id((GF_Node*)pMat), pMat, pMap->id, pMap->did);
	pLine = (M_XLineProperties*)gf_node_new(lib->pScene, TAG_MPEG4_XLineProperties);
	gf_node_register((GF_Node*)pLine, (GF_Node*)pMat);
	
	pMat->lineProps = (GF_Node*)pLine;
	pMap->pMaterial = pMat;
	// Set default values here.
	pLine->width = 1;
	ParseGCParams(lib, params, pMap);
	return 0;
	}

Table 4. Code extraction in C language from the LASeR conversion classes

XParser function	Conversion to LASeR function

parseXPolyRectangle	int X11toLASERLib_AddPolyRectangle(X11toLASERLibPtr pLib, X11toLASERLib__RectangleParams* params) {
	params->isFillRectangle = 0;
	return X11toLASERLib_AddPolyFillRectangle(pLib, params);
	}

parseXPolyFillRectangle	int X11toLASERLib_AddPolyFillRectangle(X11toLASERLibPtr pLib, X11toLASERLib__RectangleParams* params) {
	X11toLASERLib *lib = (X11toLASERLib*)pLib;
	SVG_Element *pNode;
	GCMapPtr pMap;
	int isBackgroundOrig;
	if (!(pNode = (SVG_Element*)FindDrawableLsr(lib, params->did)))
	pNode = (SVG_Element*)FindDrawableLsr(lib, "0x00000000");
	pMap = FindGCMapLaser(lib, params->gc);
	
	if (pMap){
	DEBUG_PRINTF("PFR: GC %s, pFill %p, pStroke %p\n", params->gc, pMap->pFillColor, pMap->pStrokeColor);
	isBackgroundOrig = pMap->isBackground;
	pMap->isBackground = 1;
	if (!pMap->noRects) {
	PutRectangle(lib, pNode, pMap, params);
	} else {
	int i;
	for (i = 0; i < pMap->noRects; ++i) {
	X11toLASERLib__Rectangle* paramsRect = &(params->rect);
	X11toLASERLib__Rectangle* maskRect = &(pMap->rects[i]);
	X11toLASERLib__RectangleParams newParams;
	X11toLASERLib__Rectangle* newRect = &newParams.rect;
	
	int endParamsRect = paramsRect->x + paramsRect->width;
	int endMaskRect = maskRect->x + maskRect->width;
	int endNewRect = MIN(endParamsRect, endMaskRect);
	strcpy(newParams.did, params->did);
	strcpy(newParams.gc, params->gc);
	newParams.pFillColor = params->pFillColor;
	newParams.pStrokeColor = params->pStrokeColor;
	newParams.lineWidth = params->lineWidth;
	newParams.isFillRectangle = params->isFillRectangle;
	
	newRect->x = MAX(paramsRect->x, maskRect->x);
	newRect->width = endNewRect - newRect->x;
	DEBUG_PRINTF("[X] Params: %d+%d; mask: %d+%d; result: %d+%d\n",
	paramsRect->x, paramsRect->width,
	maskRect->x, maskRect->width,

```
                                                                newRect->x,
newRect->width);
                                        endParamsRect = paramsRect->y +
paramsRect->height;
                                        endMaskRect = maskRect->y + maskRect-
>height;
                                        endNewRect = MIN(endParamsRect,
endMaskRect);
                                        newRect->y = MAX(paramsRect->y,
maskRect->y);
                                        newRect->height = endNewRect -
newRect->y;
                                        DEBUG_PRINTF("[Y] Params: %d+%d;
mask: %d+%d; result: %d+%d\n",
                                                        paramsRect->y,
paramsRect->height,
                                                        maskRect->y,
maskRect->height,
                                                        newRect->y,
newRect->height);
                                        if (newRect->width > 0 && newRect-
>height > 0)
                                                PutRectangle(lib, pNode,
pMap, &newParams);
                                }
                        }
                        pMap->isBackground = isBackgroundOrig;
                }
                X11toLASERLib_SaveMP4(lib);
                return 0;
}
```

parseXPutImage

```
int X11toLASERLib_PutImage(X11toLASERLibPtr pLib,
X11toLASERLib__ImageParams* params) {
        X11toLASERLib                   *lib = (X11toLASERLib*)pLib;
        GF_SceneGraph                   *pScene = (GF_SceneGraph*)lib-
>pScene;
        DrawableMap                             *pDrawableMap;
        SVG_Element                             *pNode;
        SVG_Element                             *pImage = NULL;
        SVG_Element                             *pUse = NULL;
        GF_FieldInfo                    h;
        GF_FieldInfo                    w;
        GF_FieldInfo                    transform;
        GF_FieldInfo                    xlink;
        GF_FieldInfo                    stroke_width;
        GCMapPtr        pMap;
        int                                     ID;
        char                                    name[15];
        unsigned int components;
        unsigned int padding = get_X11_bit_padding(params->dept) /8;
        unsigned int image_size;
        char* image;
        int videoIndex;
#ifdef USE PRUNER
        NodeIDMapPtr pMap1;
#endif

        // Only 24-bit images are supported now.
        if (params->dept != 24 && strncmp("data:", (char*)(params-
>image), 5)) {
                DEBUG_PRINTF("Unsupported image depth: %d. Ignoring
image.\n", params->dept);
                return 0;
        }

        if (!(pDrawableMap = FindDrawableMapLsr(lib, params->did)))
                pDrawableMap = FindDrawableMapLsr(lib, "0x00000000");

        pNode = pDrawableMap->pDrawable;
        pMap = FindGCMapLaser(lib, params->gc);

        videoIndex = CheckVideoTrigger(lib, params);
        if (videoIndex >= 0 && IsNodeInGraph((GF_Node*)pNode)) {
                int err = HandleVideoTrigger(lib, params, videoIndex);
                if (err || videoMap[videoIndex].url)
                        return err;
                DEBUG_PRINTF("[LASER] VideoTrigger triggered but NULL
url, "
                                        "drawing normal picture.\n");
        }

        if (strncmp("data:", (char*)(params->image), 5)) {
                // This is not a CopyArea.
```

```
                    if (params->dept <= 8)
                            components = 1;
                    else components = params->dept /8;

                    image_size = params->size / padding * components;
                    image = malloc(image_size);
                    copyPixmapX11toRGB((char*)params->image, image,
params->w, params->h, padding, components);
                    pImage = (SVG_Element*)ImageHashLaser_Find(lib-
>ImageHashLaser1, params->w, params->h, components,
                                    (unsigned char*)image);
            }
            if (!pImage) {
                    if (pMap->noRects > 0) {
                            int i;
                            DEBUG_PRINTF("PutImage: [UNSUPPORTED] found
clip rectangles for image "
                                            "%dx%d @ (%d, %d) on
drawable %s.\n", params->w, params->h,
                                            params->x, params->y,
params->did);
                            for (i = 0; i < pMap->noRects; i++) {
                            DEBUG_PRINTF("PutImage: clip rectangle %d/%d:
%dx%d @ (%d, %d).\n",
                                            i + 1, pMap->noRects,
                                            pMap->rects[i].width, pMap-
>rects[i].height,
                                            pMap->rects[i].x, pMap-
>rects[i].y);
                            }
                    }

                    pImage = (SVG_Element*)gf_node_new(pScene,
TAG_SVG_image);
                    gf_node_list_add_child(&(lib->pDefs->children),
(GF_Node*)pImage);
                    gf_node_register((GF_Node*)pImage, (GF_Node*)lib-
>pDefs);

                    gf_node_get_attribute_by_tag((GF_Node *)pImage,
TAG_SVG_ATT_width, 1, 1, &w);
                    ((SVG_Length *)w.far_ptr)->value = params->w;
                    ((SVG_Length *)w.far_ptr)->type = SVG_NUMBER_PX;

                    gf_node_get_attribute_by_tag((GF_Node *)pImage,
TAG_SVG_ATT_height,1, 1, &h);
                    ((SVG_Length *)h.far_ptr)->type = SVG_NUMBER_PX;
                    ((SVG_Length *)h.far_ptr)->value = params->h;

                    gf_node_get_attribute_by_tag((GF_Node *)pImage,
TAG_SVG_ATT_stroke_width,1, 1, &stroke_width);
                    ((SVG_Length *)stroke_width.far_ptr)->type =
SVG_NUMBER_PX;
                    ((SVG_Length *)stroke_width.far_ptr)->value = 0;

                    gf_node_get_attribute_by_tag((GF_Node *)pImage,
TAG_XLINK_ATT_href,1, 1, &xlink);
                    ((XMLRI *)xlink.far_ptr)->type = XMLRI_STRING;

                    ID = gf_sg_get_next_available_node_id(pScene);
                    sprintf(name, "DPI%d", ID);
                    gf_node_set_id((GF_Node*)pImage, ID, name);

                    if (!strncmp("data:", (char*)(params->image), 5)) {
                            abort();
                            ((XMLRI*)xlink.far_ptr)->string =
strdup((char*)params->image);
                    } else {
                            ImageHash_Insert_ID(lib->imageHash, params->w,
params->h,
                                            components,(unsigned
char*)image, (GF_Node*)pImage);
                            ((XMLRI *)xlink.far_ptr)->string =
PutBase64Image(pLib, params, image, components);

                            if (params->x == 0 && params->y == 0){
                                    pDrawableMap->pColor.red = params-
>image[2] / 255.f;
                                    pDrawableMap->pColor.green = params-
>image[1] / 255.f;
                                    pDrawableMap->pColor.blue = params-
>image[0] / 255.f;
                                    pDrawableMap->pColor.type =
```

	SVG_COLOR_RGBCOLOR;
	}
	}
	addInsertCommandLaser(pLib, (GF_Node*)lib->pDefs, (GF_Node*)pImage, -1);
	} else {
	pUse = (SVG_Element*)gf_node_new(pScene, TAG_SVG_use);
	gf_node_list_add_child(&(pNode->children), (GF_Node*)pUse);
	gf_node_register((GF_Node*)pUse, (GF_Node*)pNode);
	
	gf_node_get_attribute_by_tag((GF_Node *)pUse, TAG_SVG_ATT_transform, 1, 1, &transform);
	((SVG_Transform *)transform.far_ptr)->is_ref = SVG_TRANSFORM_MATRIX;
	gf_mx2d_add_translation(&((SVG_Transform *)transform.far_ptr)->mat,
	params->x,
	params->y);
	
	gf_node_get_attribute_by_tag((GF_Node *)pUse, TAG_XLINK_ATT_href,1, 1, &xlink);
	((XMLRI *)xlink.far_ptr)->type = XMLRI_STRING;
	
	((XMLRI*)xlink.far_ptr)->string = malloc(1 + strlen(gf_node_get_name((GF_Node*)pImage)) + 1);
	sprintf(((XMLRI*)xlink.far_ptr)->string, "#%s", gf_node_get_name((GF_Node*)pImage));
	
	ID = gf_sg_get_next_available_node_id(pScene);
	sprintf(name, "UPI%d", ID);
	gf_node_set_id((GF_Node*)pUse, ID, name);
	if (addInsertCommandLaser(pLib, (GF_Node*)pNode, (GF_Node*)pUse, -1)) {
	//save package
	lib->forceSave = 1;
	X11toLASERLib_SaveMP4(lib);
	return 0;
	}

parseXPolyText8	int X11toLASERLib_PolyText8(X11toLASERLibPtr pLib, X11toLASERLib__PolyText8Params* params) {
	X11toLASERLib *lib = (X11toLASERLib*)pLib;
	GF_SceneGraph *pScene = (GF_SceneGraph*)lib->pScene;
	SVG_Element *pFirstNode;
	SVG_Element *pNode;
	GCMapPtr pMap;
	SVG_Element *text;
	GF_FieldInfo fill;
	GF_FieldInfo gTransform;
	GF_DOMText *content;
	// M_FontStyle *pStyle;
	GF_CommandField *inf;
	GF_Command* InsertCmd;
	int ID;
	char name[15];
	pMap = FindGCMapLaser(lib, params->gc);
	pFirstNode = (SVG_Element*)FindDrawableLsr(lib, "0x00000000");
	if (!(pNode = (SVG_Element*)FindDrawableLsr(lib, params->did)))
	pNode = (SVG_Element*)pFirstNode;
	
	text = (SVG_Element*)gf_node_new(pScene, TAG_SVG_text);
	gf_node_list_add_child(&(pNode->children), (GF_Node*)text);
	gf_node_register((GF_Node*)text, (GF_Node*)pNode);
	
	gf_node_get_attribute_by_tag((GF_Node *)text, TAG_SVG_ATT_transform, 1, 1, &gTransform);
	((SVG_Transform *)gTransform.far_ptr)->is_ref = SVG_TRANSFORM_MATRIX;
	gf_mx2d_add_translation(&((SVG_Transform *)gTransform.far_ptr)->mat, params->x, params->y);
	
	content = (GF_DOMText*)gf_node_new(pScene, TAG_DOMText);
	gf_node_list_add_child(&(text->children), (GF_Node*)content);
	gf_node_register((GF_Node*)content, (GF_Node*)text);
	
	content->textContent = strdup(params->text);
	content->type = TAG_DOMText;
	if(pMap) {

	gf_node_get_attribute_by_tag((GF_Node *)text, TAG_SVG_ATT_fill,1, 1, &fill);
	((SVG_Paint *)fill.far_ptr)->type = SVG_PAINT_COLOR;
	((SVG_Paint *)fill.far_ptr)->color = *pMap->pFillColor;
	}
	if (!IsNodeInGraph((GF_Node*)pNode))
	return 0;
	
	InsertCmd = gf_sg_command_new(pScene, GF_SG_LSR_INSERT);
	InsertCmd->node = (GF_Node *)pNode;
	inf = gf_sg_command_field_new(InsertCmd);
	inf->pos = -1;
	inf->new_node = (GF_Node *)text;
	//gf_node_register((GF_Node*)inf->new_node, NULL);
	ID = gf_sg_get_next_available_node_id(pScene);
	sprintf(name, "PT%d", ID);
	gf_node_set_id((GF_Node*)inf->new_node, ID, name);
	inf->field_ptr = &inf->new_node;
	inf->fieldType = TAG_SVG_g;
	gf_list_add(lib->cmdList, InsertCmd);
	//save package
	X11toLASERLib_SaveMP4(lib);
	return 0;
	}

parseXPolySegment	int X11toLASERLib_AddPolySegment(X11toLASERLibPtr pLib, X11toLASERLib__SegmentParams* params) {
	X11toLASERLib *lib = (X11toLASERLib*)pLib;
	GF_SceneGraph *pScene = (GF_SceneGraph*)lib->pScene;
	SVG_Element *pNode;
	GCMapPtr pMap;
	SVG_Element *polySegment;
	GF_FieldInfo x1;
	GF_FieldInfo x2;
	GF_FieldInfo y1;
	GF_FieldInfo y2;
	GF_FieldInfo stroke;
	GF_FieldInfo stroke_width;
	int ID;
	char name[15];
	if (!(pNode = (SVG_Element*)FindDrawableLsr(lib, params->did)))
	pNode = (SVG_Element*)FindDrawableLsr(lib, "0x00000000");
	polySegment = (SVG_Element*)gf_node_new(pScene,TAG_SVG_line);//creates a node
	gf_node_list_add_child(&(pNode->children), (GF_Node*)polySegment);
	gf_node_register((GF_Node *)polySegment, (GF_Node *)pNode);
	
	pMap = FindGCMapLaser(lib, params->gc);
	
	if(pMap) {
	gf_node_get_attribute_by_tag((GF_Node *)polySegment, TAG_SVG_ATT_stroke, 1, 1, &stroke);
	((SVG_Paint *)stroke.far_ptr)->color = *(pMap->pStrokeColor);
	((SVG_Paint *)stroke.far_ptr)->type = SVG_PAINT_COLOR;
	printf("Stroke Color: %f %f %f\n",
	((SVG_Paint *)stroke.far_ptr)->color.red,
	((SVG_Paint *)stroke.far_ptr)->color.green,
	((SVG_Paint *)stroke.far_ptr)->color.blue);
	gf_node_get_attribute_by_tag((GF_Node *)polySegment,TAG_SVG_ATT_stroke_width,1, 1, &stroke_width);
	((SVG_Length *)stroke_width.far_ptr)->type = SVG_NUMBER_PX;
	((SVG_Length *)stroke_width.far_ptr)->value = pMap->lineWidth;
	}
	gf_node_get_attribute_by_tag((GF_Node *)polySegment,TAG_SVG_ATT_x1,1, 1, &x1);
	((SVG_Length *)x1.far_ptr)->type = SVG_NUMBER_PX;
	((SVG_Length *)x1.far_ptr)->value = params->x[0];
	gf_node_get_attribute_by_tag((GF_Node *)polySegment,TAG_SVG_ATT_x2,1, 1, &x2);
	((SVG_Length *)x2.far_ptr)->type = SVG_NUMBER_PX;
	((SVG_Length *)x2.far_ptr)->value = params->x[1];
	gf_node_get_attribute_by_tag((GF_Node

	*)polySegment,TAG_SVG_ATT_y1,1, 1, &y1);
	((SVG_Length *)y1.far_ptr)->type = SVG_NUMBER_PX;
	((SVG_Length *)y1.far_ptr)->value = params->y[0];
	gf_node_get_attribute_by_tag((GF_Node
	*)polySegment,TAG_SVG_ATT_y2,1, 1, &y2);
	((SVG_Length *)y2.far_ptr)->type = SVG_NUMBER_PX;
	((SVG_Length *)y2.far_ptr)->value = params->y[1];
	
	ID = gf_sg_get_next_available_node_id(pScene);
	sprintf(name, "PS%d", ID);
	gf_node_set_id((GF_Node*)polySegment, ID, name);
	
	addInsertCommandLaser(pLib, (GF_Node*)pNode,
	(GF_Node*)polySegment, -1);
	
	//save package
	X11toLASERLib_SaveMP4(lib);
	return 0;
	}

parseXCopyArea	int X11toLASERLib_CopyArea(X11toLASERLibPtr pLib,
	X11toLASERLib__CopyAreaParams* params) {
	X11toLASERLib *lib = (X11toLASERLib*)pLib;
	GCMapPtr pMap;
	SVG_Element *pNode_src;
	SVG_Element *pNode_dst;
	SVG_Element *pNode_cur;
	SVG_Element *pFirstMap;
	SVG_Element *pFirst;
	SVG_Element *node;
	SVG_Element *pRec;
	SVG_Element *pImage;
	GF_FieldInfo w, h, wRec, hRec, xlink, stroke,
	strokeWidth, fill;
	int count;
	int i;
	pFirst = (SVG_Element *)FindDrawableLsr(lib, "0x00000000");
	if (!(pNode_src = (SVG_Element*)FindDrawableLsr(lib, params-
	>src_did))) {
	printf("Src Draw not found : %s\n", params->src_did);
	pNode_src = (SVG_Element*)pFirst;
	}
	if (!(pNode_dst = (SVG_Element*)FindDrawableLsr(lib, params-
	>dst_did))) {
	printf("Dst Draw not found : %s\n", params->dst_did);
	// XXX(gardengh): Fix this and remove abort():
	pFirstMap is never initialized
	// before, what should be its value?
	abort();
	pNode_dst = (SVG_Element*)pFirstMap;
	}
	pNode_cur = pNode_src;
	count = gf_node_list_get_count(pNode_cur->children);
	for(i = 0; i < count; i++) {
	node = (SVG_Element*)gf_node_list_get_child(pNode_cur-
	>children, i);
	if (node){
	GF_FieldInfo transform;
	int transform_x;
	int transform_y;
	printf("CopyArea analyzing node with id %d,
	name %s, element <%s>\n",
	
	gf_node_get_id((GF_Node*)node),
	
	gf_node_get_name((GF_Node*)node),
	
	gf_node_get_class_name((GF_Node*)node));
	gf_node_get_attribute_by_tag((GF_Node*)node,
	TAG_SVG_ATT_transform, 1, 1, &transform);
	transform_x =
	(((SVG_Transform*)transform.far_ptr)->mat).m[2];
	transform_y =
	(((SVG_Transform*)transform.far_ptr)->mat).m[5];
	
	if ((transform_x < params->src_x) ||
	(transform_y < params-
	>src_y) ||
	(transform_x >= params-
	>src_x + params->w) ||
	(transform_y >= params-
	>src_y + params->h)) {
	DEBUG_PRINTF(" - Skipping node from
	CopyArea, it's out of bounds.\n");
	continue;

```
                                }
                                if(gf_node_get_tag((GF_Node*)node) ==
TAG_SVG_rect) {
                                        pRec = (SVG_Element*)node;
                                        X11toLASERLib__RectangleParams
params_rect;
                                        params_rect.isFillRectangle = 1;
                                        strcpy(params_rect.did, params-
>dst_did);
                                        strcpy(params_rect.gc, params->gc);

                                        gf_node_get_attribute_by_tag((GF_Node
*)pRec, TAG_SVG_ATT_width, 1, 1, &wRec);
                                        params_rect.rect.width = ((SVG_Length
*)wRec.far_ptr)->value;

                                        gf_node_get_attribute_by_tag((GF_Node
*)pRec, TAG_SVG_ATT_height,1, 1, &hRec);
                                        params_rect.rect.height =
((SVG_Length *)hRec.far_ptr)->value;

                                        params_rect.rect.x = params->dst_x +
transform_x - params->src_x;
                                        params_rect.rect.y = params->dst_y +
transform_y - params->src_y;

                                        printf(" + This is the CopyArea
<rect>, w = %d h = %d x = %d y = %d \n",

        params_rect.rect.width, params_rect.rect.height,
params_rect.rect.x,

        params_rect.rect.y);

                                        pMap = FindGCMapLaser(lib, params-
>gc);
                                        if (pMap){

        params_rect.pFillColor = malloc(sizeof(SVG_Color));

        params_rect.pStrokeColor = malloc(sizeof(SVG_Color));


        gf_node_get_attribute_by_tag((GF_Node *)pRec, TAG_SVG_ATT_fill,
1, 1, &fill);

        *((SVG_Color*)(params_rect.pFillColor)) = ((SVG_Paint
*)fill.far_ptr)->color;


        gf_node_get_attribute_by_tag((GF_Node *)pRec,
TAG_SVG_ATT_stroke,1, 1, &stroke);

        *((SVG_Color*)(params_rect.pStrokeColor)) = ((SVG_Paint
*)stroke.far_ptr)->color;


        gf_node_get_attribute_by_tag((GF_Node *)pRec,
TAG_SVG_ATT_stroke_width,1, 1, &strokeWidth);

        params_rect.lineWidth = ((SVG_Length *)strokeWidth.far_ptr)-
>value;

                                                        if (((SVG_Paint
*)fill.far_ptr)->type == SVG_PAINT_COLOR) {

        X11toLASERLib_AddPolyFillRectangle(pLib, &params_rect);
                                                        } else if
(((SVG_Paint *)fill.far_ptr)->type == SVG_PAINT_NONE) {

        X11toLASERLib_AddPolyRectangle(pLib, &params_rect);
                                                        } else {
                                                                abort();
                                                        }

        free(params_rect.pFillColor);

        free(params_rect.pStrokeColor);
                                        } else {
                                                abort();

        X11toLASERLib_AddPolyRectangle(pLib, &params_rect);
                                        }
                                } else if (gf_node_get_tag((GF_Node*)node) ==
```

```
TAG_SVG_image) {
#ifndef DONT_USE_USE_ELEMENT
                                        abort();
#endif
                                        printf(" + This is the CopyArea
<image>\n");
                                                pImage = (SVG_Element*)node;
                                                X11toLASERLib__ImageParams
params_image;

        strcpy(params_image.did,params->dst_did);
                                                strcpy(params_image.gc,
params->gc);

        gf_node_get_attribute_by_tag((GF_Node *)pImage,
TAG_SVG_ATT_width, 1, 1, &w);
                                                params_image.w =
((SVG_Length *)w.far_ptr)->value;

        gf_node_get_attribute_by_tag((GF_Node *)pImage,
TAG_SVG_ATT_height,1, 1, &h);
                                                params_image.h =
((SVG_Length *)h.far_ptr)->value;
                                                params_image.x = params-
>dst_x + transform_x - params->src_x;
                                                params_image.y = params-
>dst_y + transform_y - params->src_y;

        gf_node_get_attribute_by_tag((GF_Node *)pImage,
TAG_XLINK_ATT_href,1, 1, &xlink);
                                                params_image.image =
(unsigned char*)strdup(((XMLRI *)xlink.far_ptr)->string);

        X11toLASERLib_PutImage(pLib, &params_image);
                                } else if (gf_node_get_tag((GF_Node*)node) ==
TAG_SVG_use) {
                                        SVG_Element* pUse;
                                        int x;
                                        int      y;
#ifdef DONT_USE_USE_ELEMENT
                                        abort();
#endif
                                        printf(" + This is the CopyArea <use>
(image, hopefully)\n");
                                        pUse = (SVG_Element*)node;
                                        x = params->dst_x + transform_x -
params->src_x;
                                        y = params->dst_y + transform_y -
params->src_y;
                                        DEBUG_PRINTF("CopyArea USE: dst: %d,
%d; transform: %d, %d; "
                                                        "src: %d, %d;
result: %d %d\n", params->dst_x, params->dst_y,
                                                        transform_x,
transform_y, params->src_x, params->src_y, x, y);

                                        PutUpdatedUse(pLib, params->dst_did,
pUse, x, y);
                                }
                        }
        }
        return 0;
}
```

parseXCreateGC

```
int X11toLASERLib_CreateGC(X11toLASERLibPtr pLib,
X11toLASERLib__GCParams* params) {
        X11toLASERLib           *lib = (X11toLASERLib*)pLib;
        GCMapPtr                        pMap;
        pMap = (GCMapPtr)malloc(sizeof(GCMap));
        strcpy(  pMap->id, params->cid);
        strcpy(pMap->did, params->did);
        pMap->noRects = 0;
        pMap->rects = NULL;
        gf_list_add(lib->lstGC, pMap);
        pMap->pFillColor = malloc(sizeof(SVG_Color));
        memset(pMap->pFillColor, 0, sizeof(SVG_Color));
        pMap->pFillColor->type = SVG_COLOR_RGBCOLOR;
        pMap->pStrokeColor = malloc(sizeof(SVG_Color));
        memset(pMap->pStrokeColor, 0, sizeof(SVG_Color));
        pMap->pStrokeColor->type = SVG_COLOR_RGBCOLOR;
        ParseGCParamsLaser(lib, params, pMap);
        return 0;
}
```

List of publications

Journals

[1] B. Joveski, M. Mitrea, P. Simoens, I.J. Marshall, F. Prêteux, B. Dhoedt, “Semantic multimedia remote display for mobile thin clients”, Multimedia Systems, inprint; (electronic version available: DOI: 10.1007/s00530-013-0304-6, March 2013)

[2] P. Simoens, B. Joveski, L. Gardenghi, I.J. Marshall, B. Vankeirsbilck, M. Mitrea, F. Prêteux, F. De Turck, B. Dhoedt, “Optimized mobile thin clients through a MPEG-4 BiFS semantic remote display framework”, Multimedia Tools and Applications, (DOI: 10.1007/s11042-011-0849-3, August 2011), Vol:61/2, pages: 447 – 470, November 2012

Conferences

[1] R.-R. Ganji, M. Mitrea, B. Joveski, F. Prêteux, "HTML5 as an application virtualization tool", to be published in IEEE 16th International Symposium on Consumer Electronics (ISCE), June 2012, Harrisburg, PA-US

[2] B. Joveski, L. Gardenghi, M. Mitrea, F. Prêteux, “Towards collaborative MPEG-4 BiFS mobile thin remote viewer”, IEEE 15th International Symposium on Consumer Electronics (ISCE), Singapore, June 2011

[3] B. Joveski, P.Simoens, L.Gardenghi, J.Marshall, M. Mitrea, B. Vankeirsbilck, F. Prêteux, B. Dhoed, “Towards a multimedia remote viewer for mobile thin clients”, in Proceedings of SPIE, Vol: 7881, 788102 (2011); DOI:10.1117/12.876279, 2011, San Francisco, January 2011

[4] B. Joveski, M. Mitrea, F. Preteux, “MPEG-4 LASeR - based thin client remote viewer”, EUVIP2010 - European Workshop on Visual Information Processing, Paris, July 2010

[5] M. Mitrea, B. Joveski, F. Preteux, “MPEG-4 interactive image transmission on mobile thin clients” in Proceedings of SPIE Volume 7723, pp. 77230Z-77230Z-9 (2010), Brussels, April 2010

[6] M. Mitrea, P. Simoens, B. Joveski, I. J. Marshall, A. Tanguengayte, F. Preteux, B. Dhoed, “BiFS based approaches to remote display for mobile thin clients”, in Proceedings of SPIE, Vol: 7444, 74440F (2009); DOI:10.1117/12.828152, San Diego, August 2009

Patents

[1] M. Mitrea, B. Joveski, L. Gardenghi, I.J. Marshall, F. Prêteux, “Procédés et appareils de production et de traitement de représentations des scènes multimedia”, French patent

request # 1153387, April 2011 ; international patent request PCT/FR2012/050849, April 2012

[2] I.J. Marshall, M. Mitrea, B. Joveski, L. Gardenghi, F. Prêteux, "Codage de données sans perte pour communication bidirectionnelle", French patent request # 1151727, March 2011 ; international patent request PCT/FR2012/050428, March 2012

ISO MPEG contributions

[1] Mihai MITREA, Pieter SIMOENS, Bojan JOVESKI, Bert VANKEIRSBILCK, Abdeslam TAGUENGAYTE, Françoise PRETEUX, "*Novel approaches to remote display representations: BiFS-based solution and its deployment within the FP7 MobiThin project*", M16058, February 2009, Lausanne, Swiss

[2] Bojan JOVESKI, Mihai MITREA, Pieter SIMOENS, Iain-James MARSHALL, Françoise PRETEUX, "*BiFS-based solution and its deployment within the FP7 MobiThin project: event handling and streaming*", M16465, April 2009, Maui

[3] Bojan JOVESKI, Iain-James MARSHALL, Mihai MITREA, Françoise PRETEUX, Pieter SIMOENS, "*Active and reactive components for participative, distributed and collaborative BiFS scenes"*, M 16676, July 2009, London

[4] Bojan JOVESKI, Mihai MITREA, Françoise PRETEUX, Iain-James MARSHALL, Pieter SIMOENS, Bart DHOEDT, "*BiFS vs. LASeR remote display solutions for mobile thin clients"* M16882, October 2009, Xi'an

[5] Bojan JOVESKI, Iain-James MARSHALL, Mihai MITREA, Françoise PRETEUX, Pieter SIMOENS, Abdeslam TAGUENGAYTE, Bart DHOEDT *"A study of the MPEG potential for active and reactive components in participative, distributed and collaborative BiFS scenes"* M 16883, October 2009, Xi'an

[6] Bojan JOVESKI, Iain-James MARSHALL, Mihai MITREA, Françoise PRETEUX, Pieter SIMOENS, Abdeslam TAGUENGAYTE, Bart DHOEDT *"Towards adoption of collaborative BiFS technologies for use in immersive environments"* M17269, January 2010, Kyoto

[7] Bojan JOVESKI, Mihai MITREA, Françoise PRETEUX, Iain-James MARSHALL, Pieter SIMOENS, Bart DHOEDT, "*Data setup for a comparative study between BiFS & LASeR performances for thin client player"*, M 17272, January 2010, Kyoto

[8] Bojan JOVESKI, Iain-James MARSHALL, Mihai MITREA, Françoise PRETEUX "An Experimental Framework for Use of Collaborative Technology in Multi-User, Multimedia and Immersive Applications" M 17462, April 2010, Dresden

[9] Bojan JOVESKI, Iain-James MARSHALL, Mihai MITREA, Françoise PRETEUX, Pieter SIMOENS, Bart Dhoedt*"LASeR based remote viewer within the framework of FP7 MobiThin project*", M 17570, April 2010, Dresden

[10]Bojan JOVESKI, Iain-James MARSHALL, Mihai MITREA, Ludovico GARDENGHI, Françoise PRETEUX,“BiFS limitations for image compression”, M 17705, July 2010, Geneva

[11]Iain-James MARSHALL, Bojan JOVESKI, Mihai MITREA, Françoise PRETEUX,“An Experimental Framework for Use of Collaborative Technology in Multi-User, Multimedia and Immersive Applications”, M 17706, July 2010, Geneva

[12]Iain James MARSHALL, Bojan JOVESKI, Mihai MITREA, Françoise PRETEUX, “ MMT Use case for Collaborative Applications”, M17719 July 2010, Geneva,

[13]Bojan JOVESKI, Ludovico GARDENGHI, Iain-James MARSHALL, Mihai MITREA, Françoise PRETEUX,“Image transmission for mobile thin clients, comparative study”, M18422, October 2010, Guangzhou

[14]Bojan JOVESKI, Ludovico GARDENGHI, Iain-James MARSHALL, Mihai MITREA, Françoise PRETEUX, “Collaboration Technology Mandate Report on Server Command”, M19285, January 2011, Daegu

[15]Jamie MARSHALL, Mihai MITREA, Bojan JOVESKI, Ludovico GARDENGHI, Françoise PRETEUX, “Towards a Working Draft of Requirements for Collaboration Technology within MPEG”, M19287, January 2011, Daegu

[16]Jamie MARSHALL, Mihai MITREA, Bojan JOVESKI, Ludovico GARDENGHI, Françoise PRETEUX, “Draft Requirements on MPEG Scene Technology for Collaborative Applications”, M19765, March 2011, Geneva

[17]Bojan JOVESKI, Ludovico GARDENGHI, Mihai MITREA, Iain James MARSHALL, Françoise PRETEUX,“Collaboration Technology Mandate Report: Performance Issues”, M19771, March 2011, Geneva

[18]Jamie MARSHALL, Mihai MITREA, Bojan JOVESKI, Françoise Préteux, “Use Cases for Collaborative Applications”, M21161 July 2011, Torino, Italy

[19]Jamie MARSHALL, Mihai MITREA, Bojan JOVESKI, Ludovico GARDENGHI, Françoise PRETEUX , “Draft Context and Objectives for MPEG Scene Technology with Collaborative Applications”, M21162, July 2011, Torino, Italy

[20]Jamie MARSHALL, Mihai MITREA, Bojan JOVESKI, Ludovico GARDENGHI, Françoise PRETEUX,“Draft Requirements on MPEG Scene Technology for Collaborative Applications”, M21163 July 2011, Torino, Italy

[21]Bojan JOVESKI, Ludovico GARDENGHI, Mihai MITREA, Rama Rao GANJI, Iain James MARSHALL, Françoise PRETEUX, “Collaboration Technology Mandate Report: Cross-media collaboration”, M21165 Torino, July 2011, Italy

[22]Iain-James MARSHALL, Mihai MITREA, Bojan JOVESKI, Françoise PRETEUX, "Draft Call for Proposals for Scene Technologies for Collaborative Applications", M21164 Torino, July 2011, Italy

[23]Mihai MITREA, Bojan JOVESKI, Françoise PRETEUX, Iain James MARSHALL, "Collaboration Technology Mandate Report: Demonstration on Cross-Media Collaboration vs. OS peculiarities", M22704 November 2011, Geneva, Swiss

[24]Iain-James MARSHALL, Mihai MITREA, Bojan JOVESKI, Françoise PRETEUX, Draft Call for Proposals for Scene Technologies for Collaborative Applications, M22714 November 2011, Geneva, Swiss

[25]Mihai MITREA, Bojan JOVESKI, Françoise PRETEUX, Iain James MARSHALL, "Collaboration Technology Mandate Report: Demonstration on Cross-Media Collaboration vs. OS peculiarities", M22704, November 2011, Geneva, Swiss

[26]Mihai MITREA, Jamie MARSHALL, Bojan JOVESKI, Rama-Rao GANJI, Françoise PRETEUX, Youngkwon LIM, "Beyond MPEG collaboration", M23792 February 2012, San Jose, USA

[27]Mihai MITREA, Bojan JOVESKI, Iain-James MARSHALL, Françoise PRETEUX "CollaborationNode – A solution for enabling MPEG Scene Technologies for Collaborative Applications", M23983, February 2012, San Jose, USA

[28]Mihai MITREA, Bojan JOVESKI, Rama-Rao GANJI, Jamie MARSHALL, Françoise PRETEUX, Youngkwon LIM, "Beyond MPEG collaboration: MPEG-4 and HTML5", ISO/IEC JTC 1/SC 29/WG 11 M24930, May 2012, Geneva, Swiss

[29]Mihai MITREA, Bojan JOVESKI, Iain-James MARSHALL, Françoise PRETEUX, "UD Draft Requirements related to MEDUSA use case", M24931, May 2012, Geneva, Swiss

[30]Mihai MITREA, Arnaud de LA FORTELLE, Françoise PRETEUX, Bojan JOVESKI, "Car as a user - use case for User Description", M25965, July 2012, Stockholm, Sweden

[31]Mihai MITREA, Bojan JOVESKI, Iain-James MARSHALL, Françoise PRETEUX, "MEDUSA use case under the framework of the MPEG UD architecture", M25963, July 2012, Stockholm, Sweden

[32]Mihai MITREA, Arnaud de la FORTELLE, Françoise PRETEUX, Bojan JOVESKI, "MPEG-UD: car as a user", M26894, October 2012, Shanghai, China

[33]Mihai MITREA, Bojan JOVESKI, Rama-Rao GANJI, Jamie MARSHALL, Françoise PRETEUX, Youngkwon LIM, "Cross-standard collaboration: current status and perspectives", M26890, October 2012, Shanghai, China

[34]Mihai MITREA, Bojan JOVESKI, Rama-Rao GANJI, Françoise PRETEUX, Jamie MARSHALL, Youngkwon LIM, "Cross-standard collaboration: direct MPEG-4 to HTML5 communication", M26889, October 2012, Shanghai, Chin

Dispositifs de rendu distant multimédia et sémantique pour terminaux légers collaboratifs

RESUME

Cette thèse donne un aperçu critique sur les solutions existantes pour instancier les systèmes de rendu distant sur les terminaux mobiles légers (X, VNC, NX, RDP, ...). Cette confrontation entre les limites actuelles et les défis scientifiques / applicatives met en exergue que : (1) une vrai expérience multimédia collaborative ne peut pas être offerte au niveau du terminal, (2) la compression du contenu multimédia est abordée d'un seul point de vue image statique, ainsi entraînant une surconsommation des ressources réseau; (3) l'inexistence d'une solution générale, indépendante par rapport aux particularités logicielles et matérielles du terminal, ce qui représente un frein au déploiement des solutions normatives.

Cette thèse propose une architecture basée sur la gestion sémantique du contenu multimédia pour définir des systèmes de rendu distant avec fonctionnalités collaboratives.

Le principe consiste à représenter le contenu graphique généré par le serveur comme un graphe de scène multimédia interactif, enrichi avec des nouvelles composantes pour permettre la collaboration directement au niveau du contenu. Afin d'optimiser la compression du graphe de scène sous contrainte des conditions réseau variable en temps, un cadre méthodologique pour la gestion sémantique du graphe de scène a été conçu (brevet en instance). La compression des messages collaboratifs générés par les utilisateurs est réalisée grâce à un algorithme sans perte basé sur la construction dynamique, en temps réel, des dictionnaires d'encodage (solution brevetée).

Cette nouvelle architecture a été progressivement évaluée par la communauté ISO et ses nouveaux éléments collaboratifs sont actuellement acceptés comme des extensions à la norme CEI JTC 1 SC 29 ISO WG 11 (MPEG 4 BIFS). Le démonstrateur logiciel sous-jacent, dénommé MASC (Multimédia Adaptive Sémantique Collaboration) est implanté par une approche logiciel libre. MASC a été comparé à des solutions fournies par des industriels comme VNC (RFB) ou Microsoft (RDP).

Il a été démontré que: (1) MASC offre une haute qualité visuelle (PSNR compris entre 30 et 42 dB et SSIM supérieur à 0,9999), (2) la consumation de la bande passante *downlink* présente un gain de 2 à 60, tandis que la consumation de la bande passante *uplink* comporte un gain de 3 à 10, (3) la latence dans la transmission des événements générés par l'utilisateur est réduite par un facteur de 4 à 6; (4) la consumation des ressources de calcul côté client, bien que plus grande que dans le cas RDP, est réduite par un facteur de 1,5 par rapport à la VNC RFB.

Cette thèse propose la première architecture logicielle pour un système de rendu distant, basée sur (1) la gestion sémantique du contenu multimédia pour assurer une optimisation conjointe de l'utilisation du réseau et des ressources calcul côté terminal et (2) des nouveaux éléments de traitement des données de collaboration directement au niveau du contenu multimédia, pour assurer des solutions normatives en logiciel libre.

Mots-clés: terminaux légers mobile, rendu distant, contenu multimédia collaboratif, MPEG-4 BiFS & LASeR

MINES ParisTech

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY