PROX

A P2P Programming Framework

FINAL PROJECT REPORT

Submitted by

ANASWARA N V

ANJANA DAVIS

NIKHIL T NAIR

NIMI P BABY

NIRMAL KUMAR V A

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY (B.TECH)

in

COMPUTER SCIENCE & ENGINEERING

of

UNIVERSITY OF CALICUT

Under the guidance of

Mr. VIJU SHANKAR

JUNE 2012

Department of Computer Science & Engineering

JYOTHI ENGINEERING COLLEGE, CHERUTHURUTHY

THRISSUR 679 531

PROX

A P2P Programming Framework

FINAL PROJECT REPORT

Submitted by

ANASWARA N V

ANJANA DAVIS

NIKHIL T NAIR

NIMI P BABY

NIRMAL KUMAR V A

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY (B.TECH)

in

COMPUTER SCIENCE & ENGINEERING

of

UNIVERSITY OF CALICUT

Under the guidance of

Mr. VIJU SHANKAR

JUNE 2012

Department of Computer Science & Engineering

JYOTHI ENGINEERING COLLEGE, CHERUTHURUTHY

THRISSUR 679 531

Department of Computer Science & Engineering
JYOTHI ENGINEERING COLLEGE, CHERUTHURUTHY
THRISSUR 679 531

JUNE 2012

BONAFIDE CERTIFICATE

Certified that this project report **PROX - A P2P Programming Framework** being submitted in partial fulfillment of the requirements for the award of degree of **Bachelor of Technology** of **University of Calicut** is the bonafide work of **ANASWARA N V , ANJANA DAVIS , NIKHIL T NAIR , NIMI P BABY , NIRMAL KUMAR V A**, who carried out the project work under our supervision.

Prof.ASWATHY WILSON	**Mr. VIJU SHANKAR**
HOD	Lecturer
Dept.of CSE	Dept.of CSE

Acknowledgement

We take this opportunity to express our heartfelt gratitude to all respected personalities who had guided, inspired and helped us in the successful completion of this project.

First and foremost, we express our thanks to **The Lord Almighty** for guiding us in this endeavour and making it a success.

We are thankful to our Principal **Dr.Gylson Thomas** and the Management for providing us with excellent lab and infrastructure facilities.

Our sincere thanks to the Head of the Department of Computer Science And Engineering, **Prof. Aswathy Wilson** for her valuable guidance and suggestions.

We would like to express our deepest gratitude to **Mr. Viju Shankar** for his valuable contributions and guidance.

Last but not least, we thank all our teaching and non teaching staffs of Department of Computer Science And Engineering, and also our friends for their immense support and help in all the stages for the development of the project.

Abstract

Application developed using the current programming platforms executes on a single system, when the application too large for example (image analysing, large data handling, games, HD video playback system) needs more computational space and CPU utilizations to work and perform better. If the user system had low configuration applications fails to perform. In a network scenario in colleges, schools, companies more systems are connected in network till there applications need to be installed on every system and run.

To improve the large scale application performance, data management, accessibility we propose a new kind of programming paradigm through which application utilizes the hardwares of nearby system to execute the part of the programmes using direct connection (Peer to Peer). This programming model provides API for .NET Framework programmers to leverage upon and develop their application to get better performance out from their applications.

This system automatically finds the nearby system from the local network or in internet (in the case of public domains) and executes the application tasks on the one or more system. Application execution is scheduled automatically with maintaining the executed program part, executed data part in a stack of task memory on the application system. Results from the other computers are transported back to the application running system results are added to the application. Task execution is carried out asynchronously that improves the response time of the application. On private domains their network systems are utilized (local or other unit lab through internet). On Public domain each user is assigned an unique id and their friends who allowed the user to share the system hardware resource is used for application domain through this one can get more performing application even their system are of low hardware configuration.

CONTENTS

List of Abbreviations

P2P	: *Peer to Peer*
3D	: *Three Dimensional*

List of Figures

List of Tables

CHAPTER 1
INTRODUCTION

1.1 Overview

Basic aim of this project is to create a Peer to Peer content and application streaming media player which increases the efficiency of the video playback for DVD, High Definition Video Content and allow the users to stream the video or music files directly from their computer hard disk to anyone without copying or downloading and enabling them to user their hardware resources.

1.2 Motivation and Technical Relevance

Currently media players utilizes the users desktop system resources execute tasks, for HD video content playback media player needs more system process if the user system doesnt contains graphics card(GPU), HD playback is slower and also current media content streaming schemes adopt the client-server model for content delivery. However, such architecture is difficult to scale as media streaming is both data and processing-intensive. Prohibitively vast amount of server-side bandwidth and CPU power are required for a massive audience. On the other hand, highly scalable yet affordable computing and content sharing systems have been built successfully with peer-to-peer (P2P) networks. In both live and on-demand media streaming, content is often sent linearly after a starting point, whereas access to Media content is rather arbitrary and non linear, and depends much on real-time user behaviours.

To improve the large scale application performance, data management, accessibility we propose a new kind of programming para diagram through which application utilizes the hardwares of nearby system to execute the part of the programmers using direct connection (Peer to Peer). This programming model provides API for .NET Framework programmers to leverage upon and develop their application to get better performance out from their applications.

1.3 Progress of project

This project is done in three stages.The main and first stage is the literature survey.It is the stage were analysis of existing systems are done.We analysed various features of those systems and tried to add more features in the proposed system.The second stage includes Architecture designing and the initial prototype creation.Third stage is the final implementation of project.

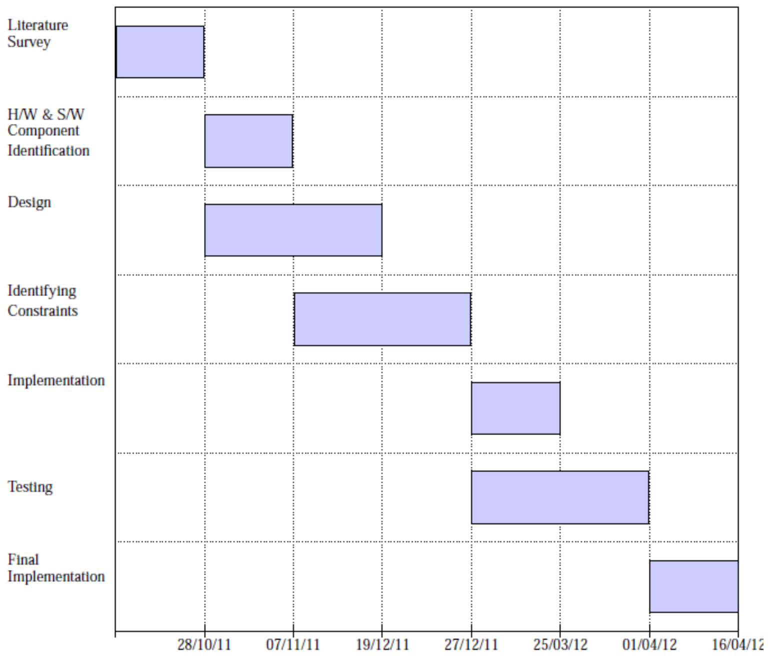


Fig. 1.1: Progress of Project

1.4 Member roles and responsibilities

Table. 1.1: Team Organization

Name	**Role/Responsibility**
ANASWARA N V	Designer
ANJANA DAVIS	Designer
NIKHIL T NAIR	Debugger
NIMI P BABY	Programmer
NIRMAL KUMAR V A	Programmer

1.5 Layout

This section provides an overview of the chapters present in this project report.

Chapter 2 presents the related software referenced during the initial survey of the project.

Chapter 3 presents the proposed system.The proposed system is intended to create a Peer to Peer content and application streaming media player which increases the efficiency of the video playback for DVD, High Definition Video Content and allow the users to stream the video or music files directly from their computer hard disk to anyone without copying or downloading and enabling them to user their hardware resources.

Chapter 4 includes the hardware and software requirements for the project.

Chapter 5 gives an overview of the schedule of the project work.It shows the module division and work effort and module allocation to each member.This chapter provides a general architecture of our project including the input design and output design.It also shows the Data Flow Diagrams(DFD) of the entire project.

Chapter 6 includes the algorithms or program code elements for the initial model (working implementation) of the project.

Chapter 7 includes the details of the unit tests, integration tests and proposals for future maintenance.

The last chapter, Chapter 8 summarizes the work done in this project.

CHAPTER 2
LITERATURE SURVEY

2.1 Related Papers

We have referenced various papers related to this concepts.4 papers were chosen as base papers. They are described below.

2.1.1 Parallel Computing

The idea of a single - processor computer is fast becoming archaic and quaint. We now have to adjust our strategies when it comes to computing

- It is impossible to improve computer performance using a single processor. Such processor would consume unacceptable power. It is more practical to use many simple processors to attain the desired performance using perhaps thousands of such simple computers .
- As a result of the above observation, if an application is not running fast on a single - processor machine, it will run even slower on new machines unless it takes advantage of parallel processing.
- Programming tools that can detect parallelism in a given algorithm have to be developed. An algorithm can show regular dependence among its variables or that dependence could be irregular. In either case, there is room for speeding up the algorithm execution provided that some subtasks can run concurrently while maintaining the correctness of execution can be assured.
- Optimizing future computer performance will hinge on good parallel programming at all levels: algorithms, program development, operating system, compiler, and hardware.
- The benefits of parallel computing [1]need to take into consideration the number of processors being deployed as well as the communication overhead of processor - to - processor and processor - to - memory. Compute - bound problems are ones wherein potential speed up depends on the speed of execution of the algorithm by the processors.

Communication - bound problems are ones wherein potential speed up depends on the speed of supplying the data to and extracting the data from the processors.

- Memory systems are still much slower than processors and their bandwidth is limited also to one word per read/write cycle.

Parallel Programming Model

Exist as an abstraction above hardware and memory architectures.Examples are as follows.

- Shared Memory
- Threads
- Messaging Passing
- Data Parallel

Here we use thread concepts. A single process may have multiple, concurrent execution paths.These are typically used with a shared memory architecture.Programmer is responsible for determining all parallelism.Threads are used because the benefits of them are Increased Performance and better resource utilization.But they have some risks.They Increase complexity of application.And they are Difficult to debug(in race condition ,deadlock etc).

2.1.2 P2P 3D Media Streaming

Virtual worlds have become very popular in recent years, with trends towards larger worlds and more user-generated content. The growth of 3D content in virtual worlds will make real-time content streaming (or 3D streaming) increasingly attractive for developers. To meet the demands of a large user base while lowering costs, peer-to-peer (P2P) content delivery holds the promise for a paradigm shift in how future virtual worlds will be deployed and used. Here define both the problem and solution spaces for P2P 3D streaming, by outlining its requirements, challenges, and categorizing existing proposals. Open questions are also identified to facilitate the design of future systems.

Current Media streaming schemes adopt the client-server model for content delivery. However, such architecture is difficult to scale as 3D streaming [2]is both data- and processing-intensive. Prohibitively vast amount of server-side bandwidth and CPU power are required for a massive audience. On the other hand, highly scalable yet affordable computing and content sharing systems have been built successfully with peer-to-peer (P2P) networks.

Following are some features we adopted from 3D streaming concepts.

- Peer Discovery:

  In order to know which peers media need to be streamed, the user client must first discover the peers within its Peer Cloud. As only the server initially possesses complete knowledge of the peer location, we need to partition and distribute the media descriptions (i.e., object meta-data such as file name, frame details), so that a distributed discovery can be done (e.g., via hierarchical trees).

- Media Source Discovery:

  To obtain content from other user clients instead of the server, each client must know some other clients (i.e., peers) who may hold the content of interest. These partner peers likely are within each others Shared Group as overlapped visibility indicates shared interests. However, other peers who have been in the same area previously may also retain content in cache. The contents sources are maintained and discovered in distributed manner.

- Media State Exchange:

  Once a few peers are found, we still need to know which content pieces are available at which peers, and what network conditions exist for each peers, so that content requests can be fulfilled. A nave approach is to query each known peer. However, the query-response time of such a pull approach may not meet the real-time requirements of media streaming. A push approach is followed for peers to pro actively notify each other of content availability.

- Media Content Exchange:

  To optimize the visual (or streaming) quality for a given bandwidth budget, a client can then leverage its knowledge on peers to schedule content requests based on visibility, content availabilities and network conditions. As long as the piece dependencies are satisfied, only a roughly sequential transfer is needed (as opposed to the strictly sequential video streaming). Concurrent downloads can also be exploited to accelerate the retrieval for pieces that do not involve dependencies. Depending on the results of initial requests, additional peers or requests may then be needed.

- Partition:

  The task of dividing the entire scene into smaller units so that global knowledge of all object placements is not required for visibility determination. Media partition is essential if visibility calculations were to be decentralized.

- Fragmentation:

  The task of dividing Media Files into pieces so that they may be transmitted via a network and reconstructed back progressively by a client.

- Prefetching:

  The task of predicting data usage ahead of time and generating data requests so that transmission latency is masked from users. Predications of media behaviours can be employed for this task.

- Prioritization:

  The task of performing visibility determination to generate the request order for object pieces. The goal is to maximize visual quality by considering factors such as distance, line-of-sight, or visual importance.

- Selection:

  The task of determining the proper peers to connect and pieces to obtain based on considerations of peer capacity, content availability and network conditions, in order to efficiently fulfil pre fetching and prioritization needs.

2.1.3 Peer-to-peer media streaming application survey

Up to now, the Internet has been growing both in size and technologically. It is also not uncommon to find many types of media within a single web-page such as 3D animations, sound, video clips, online radio and TV services. Streaming is the technology which enables this, by viewing a media while downloading it. An important amount of research [3] has been achieved to improve this bandwidth sensitive technology. Peer-To-Peer (P2P) network is at the moment one of the most effective solution to distribute streaming data within a large scale overlay, involving potentially thousands of nodes.

A few methods of the peer to peer media streaming technology can be adapted while building the PROX. The paper reviewed is going through different media streaming techniques.

Cool Streaming

Cool Streaming is a data-driven overlay network for P2P live media streaming implemented by the Universities of Hong-Kong and Vancouver. This application coded in Python language creates its own overlay P2P network following a mesh topology. Using an efficient scheduling algorithm to fetch video segments from each peer and a strong buffering system, Cool Streaming achieves a smooth video playback and a very good scalability as well as performance.Cool Streaming does not follow some kind of distribution structure to deliver the media, but bases its delivery on data-availability. Media streams are split in segments of fixed size. Each peer owns a buffer map of 120 bits: which reflects a 120 segments sliding window. During a streaming session, peers continuously exchange their buffer map, which describes what data is available. Each peer downloads data it needs at any given time.

Cool Streaming is a good example of live media broadcast application. It presents an interesting approach close to the Bit Torrent concept . The algorithm to distribute the data is simple, scalable and totally distributed. By adopting a mesh topology and a data-centric approach, Cool Streaming system is less sensitive to peer failures than streaming systems based on tree overlays.

End System Multicast

End System Multicast (ESM) is a complete infrastructure for media broadcasting, implemented by Carnegie Mellon University. ESM allows to broadcast audio/video data to a large pool of users. The information is delivered following a traditional single-tree approach, which implies that any given peer receives streams from only one source. They also support a contributor-aware policy rather than a first-come-first-served approach. In a contributor-aware policy, the system knows which peers participate actively in the network based on resource availability or processing time. A contributor (peer which can get children) is then assigned more resources over a free-rider (peer who does not accept any child within the distribution tree or which is not so active).

ESM is one of the pioneer of P2P media streaming and their goal is mainly to prove that application-level overlay multicast is a possible solution. Their system is based on a single tree overlay which make it highly sensitive to peer failures or disconnection.

Gnu stream

Gnu Stream is a receiver-driven P2P multimedia streaming implemented by Purdue University. It supports streaming from multiple sources by performing load balancing between sender peers. The allocation is either even or proportional, where in the first case all peers send the same amount of data and in the other case peers send as much data as they can.

Similar to Cool Streaming, Gnu Stream uses a layered architecture to build its solution. Dividing the system in three layers; one to communicate with the network, one for streaming and one for the display, seems to be a good approach, which ensures clarity and portability of the software.

Peer Streaming

Peer Streaming is a Microsoft receiver-driven P2P media streaming system. The general architecture follows a client/server scheme and the P2P network helps the server in distributing the media content.Peer Streaming project provides useful guidelines for P2P media streaming. Load-balancing between sender peers could be a good solution to enable streaming from multiple sources. Similar to Gnu Stream, a client requests data from multiple peers, a provider sends as much data as it can at any given time, resulting in a proportional allocation of the streaming load.

In the above discussed streaming techniques, Cool streaming and End System Multicast are two of the leading techniques. The End System Multicast uses the tree structure, although give high video quality it is highly sensitive to peer failures or disconnection. Cool streaming is based on mesh topology and has less peer failures than that based on tree overlays. Here each peer downloads data it needs at any given time. So found out that for PROX an optimum approach would be using the mesh structure.

2.1.4 Engineering the Cloud from Software Modules

Cloud Computing is used to refer to many different technologies: from outsourcing of information processing to the usage of external data center. Existing cloud services range from virtual server environments such as AMAZON EC2 to mash-up platforms for external services such as YAHOO PIPES. Regardless of the specific definition used, the underlying software fabric for cloud computing is not new: cloud computing is typically implemented as a distributed or parallel application running on a virtual cluster of computers.

Our vision for cloud computing software [4] is to provide the means to build cloud applications ignoring the fact that the final deployment will be distributed. To do so, we propose to use the widely accepted notion of software module as the unit of management and distribution and let a module management platform deal with the complexity of distributed deployment, execution, and maintenance.

Modules are units of encapsulation. Separating the code into modules encourages the developer to define interfaces and thereby to shape coupling and cohesion of the code. Cohesion means that all the functions of a module should ideally be closely related so that the resulting module encapsulates common functionality. This functionality is ideally only exported to other modules through a narrow set of interfaces to avoid coupling. Coupling means that one module depends on internal implementation details of a second module. Tight coupling is not only a burden for code evolution but also for reusing modules in a different application context. It also makes distributed deployment and maintenance very complex and unmanageable for large systems.

Designing modular applications is easier than designing distributed applications. The design principles of modules are much better understood and are by now common programming practice. However, structuring the code base into modules does not automatically make a well-structured system. Module systems which require explicit statement of dependencies make the design process more manageable and controllable.Among the many and different module approaches, OSGi for the Java language is one of the most popular and widely used systems.

2.1.5 Dynamic Load Balancing and Job Replication

Here dynamic load balancing & job replication is discussed. From this paper some features are adopted.

- PROX system have reputation based capacity scheduling
- In dynamic load balancing, it divide jobs into sub problems
- According to the scheduler, it is optimising the load & distribute the load
- In job replication, make same copies of the job & distribute them to different processors.
- Make use of threshold & statistics for conclusion.
- distribution of the load across global grids
- Finding exact number of replicas.
- More mathematical techniques have to be implemented for having high supports.

For overcome this, PROX uses reputation based capacity scheduling.

2.2 Background Study

IDEs are designed to maximize programmer productivity by providing tightly-knit components with similar user interfaces. This should mean that the programmer has much less mode switching to do than when using discrete development programs. However, because an IDE is by its very nature a complicated piece of software, this high productivity only occurs after a lengthy learning process.

Typically an IDE is dedicated to a specific programming language, so as to provide a feature set which most closely matches the programming paradigms of the language. However, some multiple-language IDEs are in use, IDEs typically present a single program in which all development is done. This program typically provides many features for authoring, modifying, compiling, deploying and debugging software. The aim is to abstract the configuration necessary to piece together command line utilities in a cohesive unit, which theoretically reduces the time to learn a language, and increases developer productivity. It is also thought that the tight integration of development tasks can further increase productivity. For example, code can be compiled while being written, providing instant feedback on syntax errors. While most modern IDEs are graphical, IDEs in use before the advent of windowing systems (such as Microsoft Windows or X11) were text-based, using function keys or hotkeys to perform various tasks (Turbo Pascal is a common example). This contrasts with software development using unrelated tools, such as VI, GCC or make.

Visual Studio, like any other IDE, includes a code editor that supports syntax highlighting and code completion using IntelliSense for not only variables, functions and methods but also language constructs like loops and queries. IntelliSense is supported for the included languages, as well as for XML and for Cascading Style Sheets and JavaScript when developing web sites and web applications

The concept of PROX is cloud, which works for computer programmers; they need some sort of code editor to do their work. PROX allows you to write code on multiple projects came together they could produce a solution for a Cloud programming atmosphere. It supports C#, PHP, VB.NET, other popular codes.

Common Language Specification(CLS) compliance generally refers to the claim that CLS rules and restrictions are being followed. However, the concept has a more specific meaning depending on whether you are describing CLS-compliant code or CLS-compliant development tools, such as a compiler. CLS-compliant tools can help you write CLS-compliant code.

The managed execution process includes the following steps, which are discussed in detail later in this topic.

2.2.1 Choosing a Compiler

To obtain the benefits provided by the common language runtime(CLR), you must use one or more language compilers that target the runtime, such as Visual Basic, C#, Visual C++, F#, or one of many third-party compilers such as an Eiffel, Perl, or COBOL compiler.

Because it is a Multilanguage execution environment, the runtime supports a wide variety of data types and language features. The language compiler you use determines which runtime features are available, and you design your code using those features. Your compiler, not the runtime, establishes the syntax your code must use. If your component must be completely usable by components written in other languages, your component's exported types must expose only language features that are included in the Common Language Specification (CLS). You can use the CLSCompliantAttribute attribute to ensure that your code is CLS-compliant. For more information, see Writing CLS-Compliant Code.

2.2.2 Compiling to MSIL

When compiling to managed code, the compiler translates your source code into Microsoft intermediate language (MSIL), which is a CPU-independent set of instructions that can be efficiently converted to native code. MSIL includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations. Before code can be run, MSIL must be converted to CPU-specific code, usually by a just-in-time (JIT) compiler. Because the common language runtime supplies one or more JIT compilers for each computer architecture it supports, the same set of MSIL can be JIT-compiled and run on any supported architecture.

When a compiler produces MSIL, it also produces metadata. Metadata describes the types in your code, including the definition of each type, the signatures of each type's members, the members that your code references, and other data that the runtime uses at execution time. The MSIL and metadata are contained in a portable executable (PE) file that is based on and that extends the published Microsoft PE and common object file format (COFF) used historically for executable content. This file format, which accommodates MSIL or native code as well as metadata, enables the operating system to recognize common language runtime images. The presence of metadata in the file together with MSIL enables your code to describe itself, which means that there is no need for type libraries or Interface Definition Language (IDL). The runtime locates and extracts the metadata from the file as needed during execution.

2.2.3 Compiling MSIL to Native Code

Before you can run Microsoft intermediate language (MSIL), it must be compiled against the common language runtime to native code for the target machine architecture. The .NET Framework provides two ways to perform this conversion.

2.2.4 Compilation by the JIT Compiler

JIT compilation converts MSIL to native code on demand at application run time, when the contents of an assembly are loaded and executed. Because the common language runtime supplies a JIT compiler for each supported CPU architecture, developers can build a set of MSIL assemblies that can be JIT-compiled and run on different computers with different machine architectures. However, if your managed code calls platform-specific native APIs or a platform-specific class library, it will run only on that operating system.

JIT compilation takes into account the possibility that some code might never be called during execution. Instead of using time and memory to convert all the MSIL in a PE file to native code, it converts the MSIL as needed during execution and stores the resulting native code in memory so that it is accessible for subsequent calls in the context of that process. The loader creates and attaches a stub to each method in a type when the type is loaded and initialized. When a method is called for the first time, the stub passes control to the JIT compiler, which converts the MSIL for that method into native code and modifies the stub to point directly to the generated native code. Therefore, subsequent calls to the JIT-compiled method go directly to the native code.

Because the JIT compiler converts an assembly's MSIL to native code when individual methods defined in that assembly are called, it affects performance adversely at run time. In most cases, that diminished performance is acceptable. More importantly, the code generated by the JIT compiler is bound to the process that triggered the compilation. It cannot be shared across multiple processes. To allow the generated code to be shared across multiple invocations of an application or across multiple processes that share a set of assemblies, the common language runtime supports an ahead-of-time compilation mode. This ahead-of-time compilation mode uses the Ngen.exe (Native Image Generator) to convert MSIL assemblies to native code much like the JIT compiler does. However, the operation of Ngen.exe differs from that of the JIT compiler in three ways:

- It performs the conversion from MSIL to native code before running the application instead of while the application is running.
- It compiles an entire assembly at a time, instead of one method at a time.
- It persists the generated code in the Native Image Cache as a file on disk.

2.2.5 Code Verification

As part of its compilation to native code, the MSIL code must pass a verification process unless an administrator has established a security policy that allows the code to bypass verification. Verification examines MSIL and metadata to find out whether the code is type safe, which means that it accesses only the memory locations it is authorized to access. Type safety helps isolate objects from each other and helps protect them from inadvertent or malicious corruption. It also provides assurance that security restrictions on code can be reliably enforced.

The runtime relies on the fact that the following statements are true for code that is verifiably type safe:

- A reference to a type is strictly compatible with the type being referenced.
- Only appropriately defined operations are invoked on an object. Identities are what they claim to be.

During the verification process, MSIL code is examined in an attempt to confirm that the code can access memory locations and call methods only through properly defined types. For example, code cannot allow an object's fields to be accessed in a manner that allows memory locations to be overrun. Additionally, verification inspects code to determine whether the MSIL has been correctly generated, because incorrect MSIL can lead to a violation of the type safety rules. The verification process passes a well-defined set of type-safe code, and it passes only code that is type safe. However, some type-safe code might not pass verification because of some limitations of the verification process, and some languages, by design, do not produce verifiably type-safe code. If type-safe code is required by the security policy but the code does not pass verification, an exception is thrown when the code is run.

CHAPTER 3
PROX SYSTEM

This System addresses media object discovery by using grid-based file descriptions and queries for neighbours from a P2P overlay network for source discovery. A query-response approach is used for state exchange, and random selection from peers for content exchange. Though this basic design is simple, the data sources are limited and the query for content availability may be slow. In a subsequent work, we let clients keep historic Media neighbours as extra sources, and pro actively push content availability to media neighbours to reduce the query-response delay. Piece requests are also sent to closer, media neighbours first to avoid concentrating requests. These are the various features of the PROX system.

- Peer Media Selection:

  A few peer selection strategies based on object proximity and estimated content availability are also evaluated. Object discovery thus is based on a distributed tree, while source discovery is performed with a P2P overlay network similar to this system. However, as only a selected set of connectivity peers provides the media neighbours, the older methods are more of a super-peer than a fully distributed design. To learn of the client states, queries and responses are also exchanged among the peers. Content availability is not exchanged, but inferred from the relative positions between neighbours. Based on response time and estimated content availability, requests are then made randomly to potential sources.

- Hyper Verse:

  Hyper Verse utilizes a group of public servers to construct a static, Structured overlay that maintains the user details. The clients learn of other peers from the servers, and exchange content by forming a loosely-structured overlay. Media discovery and source discovery thus are performed centrally by the server that notifies clients of relevant peers and objects. There is no explicit state exchange policy, while content is also requested from random neighbors. We compare these designs with this system in a taxonomy based on the main challenges, while outlining the potential solution space for P2P Media streaming, while Hyper Verse uses the server to maintain the object list. A central list is arguably more flexible and secure, as distributed media descriptions are not straightforward to update, and malicious clients could manipulate the object list. On the other hand, a central list faces scalability limitations if the server receives too many requests. Grid partitioning is simple and only a small number of cells are needed for ground-level navigation. However, for media viewable from different peers grids become inefficient as too many cells can be involved. On the other hand tree structures require a top-down construction; so many nodes from the root down have to be retrieved first, before ground-level objects can be determined.

- Source Discovery:

  The current designs mostly use Media neighbours as sources, maintained either centrally or among the peers. Using a P2P overlay network for neighbour discovery can drastically reduce server loads. However, the overlay incurs some overhead that grows with Media neighbour density. A super-peer-based tracker for sources may be a balance between the two extremes, and could keep track of non-media neighbours with relevant content.

- State Exchange:

  Few states are being exchanged among the current designs (e.g., content availability and network condition), and the two main approaches are pull (i.e., query-response) and push (i.e., proactive update). Current evaluation indicates that the push approach is faster than pull.

- Content Exchange:

  Random selection is being used by both the basic system design and Hyper Verse. In the enhanced P2P Media Streaming, requests are sent to closer neighbours first, while in other systems bases requests on estimated capacities. However, detailed comparisons have yet been made among these approaches. Additional criteria may also be used to form the requests, such latency or piece dependency. Current methods are mostly pull-based (i.e., requests are sent to the source providers), but push-based approaches is also a possibility (i.e., sources pro actively send out content, similar to how Content Delivery Networks, or CDNs push web content to different geographic servers). Besides these network-specific issues, additional requirements in Media streaming are also worth exploring. For example, performing content update with distributed peers, or providing content authentication for commercial applications. Content pre fetching and caching is also an important aspect to Media streaming.

Following Fig. 3.1 shows the block diagram for the proposed PROX System

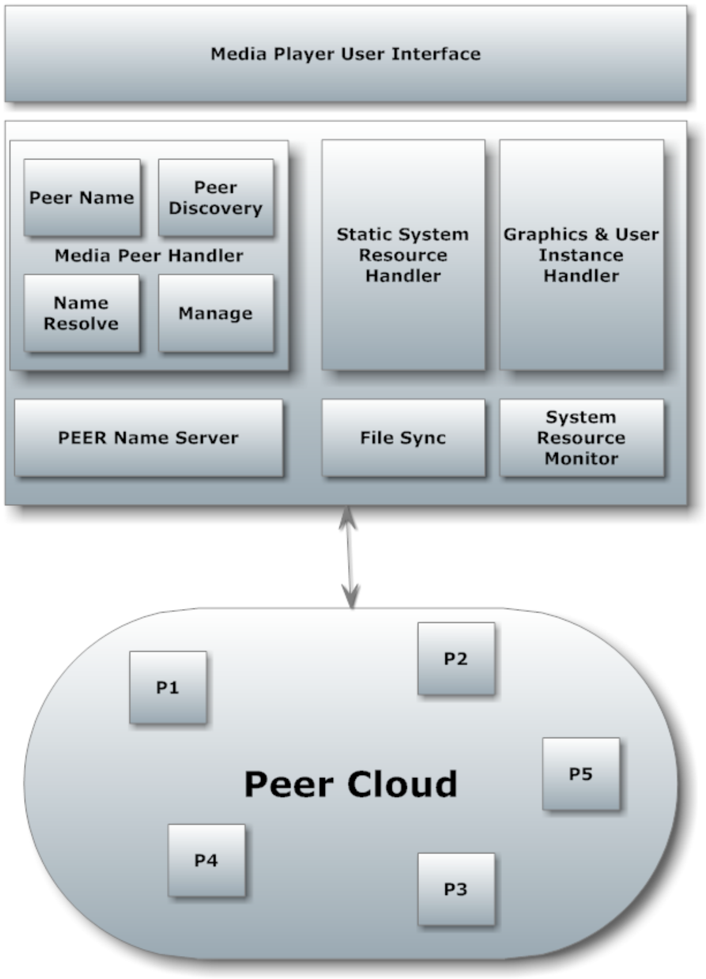


Fig. 3.1: Block Diagram of PROX System

System contains a media player user interface,which act as an interface to the system.It contains a media peer handler,which handles the peer related tasks such as managing,resolving etc.There is a peer name server which controls all peers.

CHAPTER 4
SYSTEM REQUIREMENTS SPECIFICATION

4.1 Hardware Requirements

1. Processor : Pentium IV
2. Processor Speed : 1.7 GHz
3. Memory(RAM) : 256 MB
4. Hard Disk : 80 GB

4.2 Software Requirements

- Operating System : Windows 2000 Onwards
- Software Tools : C# Windows Application, Microsoft Visual Studio.
- Database : XML

4.3 Conclusion

We use C# as the language for programming.Microsoft Visual Studio is the main software used.

CHAPTER 5
DESIGN & ANALYSIS

5.1 System Analysis

This System addresses media object discovery by using grid-based file descriptions and queries for neighbours from a P2P overlay network for source discovery. A query-response approach is used for state exchange, and random selection from peers for content exchange. Though this basic design is simple, the data sources are limited and the query for content availability may be slow. In a subsequent work, we let clients keep historic Media neighbours as extra sources, and pro actively push content availability to media neighbours to reduce the query-response delay. Piece requests are also sent to closer, media neighbours first to avoid concentrating requests.

5.1.1 Module breakup

Table. 5.1: Module Description

Module	**Description**
Registration	Includes peer Registration
Discovery	Includes cloud creation and peer discovery
Scheduling	Partition,fragmentation,pre fetching,prioritization
Application Program	Selection and runtime

Module division is done as listed above.Discovery modules include peer discovery and module source discovery.Media exchange handles state and content exchange.Scheduling is the important part.This includes partition,fragmentation,Pre fetching and prioritization.Last modules include selection and runtime.

5.1.2 Member effort

Table. 5.2: Module Allocation

#	Task	Estimated Effort	Start Date	End Date	Person
1	Registration	20-25 hrs	(08/03/2012)	(08/04/2012)	Anaswara N V
2	Discovery	10-15 hrs	(08/03/2012)	(01/04/2012)	Nimi P. Baby
3	Scheduling	20-22 hrs	(09/03/2012)	(02/04/2012)	Anjana Davis
4	Scheduling	20-22 hrs	(09/03/2012)	(02/04/2012)	Nikhil T.Nair
5	Application	15-20 hrs	(08/03/2012)	(03/04/2012)	Nirmal Kumar V A

5.2 System Design

5.2.1 Architecture

Input Design

Input Design is the process of converting a user-oriented description of the input to a computer based system into a programmer oriented specification.During the input design,analysts consults the data flow diagrams developed during the system analysis and determines the source and methods for collecting data and getting them into the system.Depending on the nature of the application an organization may enter data into its computer.

In the design phase,the complete input data are identified and the input media is selected.The data entry screen is designed so that the redundancy of data is avoided and the number of keystrokes are minimized.Online-entry makes use of a processor that accepts commands and data from the operator through a keyboard.The processor analyses the input received.It is then accepted or rejected or further input is in the form of a message displayed on the screen. Points to be noted while designing the input screen are:

- Allow only the informations that the user needs.
- Do not overcrowd the input screen.
- Keep the same style between screens.It helps the user if the screen keep the same style throughout the system.

- Ask for confirmation of critical data.
- On input,validate the data as soon as possible to prevent incorrect data to flow into the system.

Output Design

The output design is an ongoing activity that follows the beginning of the project.During the first stage of output design,analyst determines what the application produces and how to organise and present that data.The output design is the continuation of the output description in the design phase.

Outputs are the most important and direct source in information to users and to the management.Intelligent output design will improve the system relationship with the user and help in decision making.Outputs are obtained in the form of response to the request of the user.

5.2.2 Data Flow Diagrams

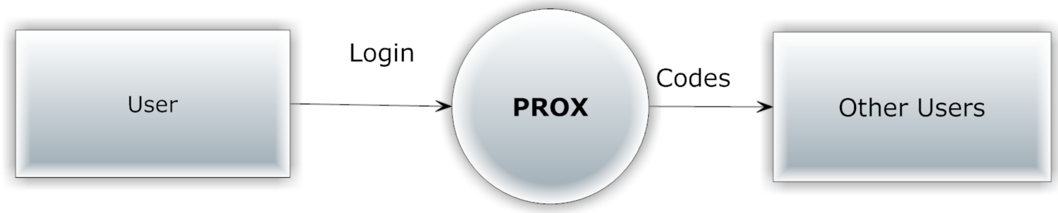


Fig. 5.1: Requesting Resources

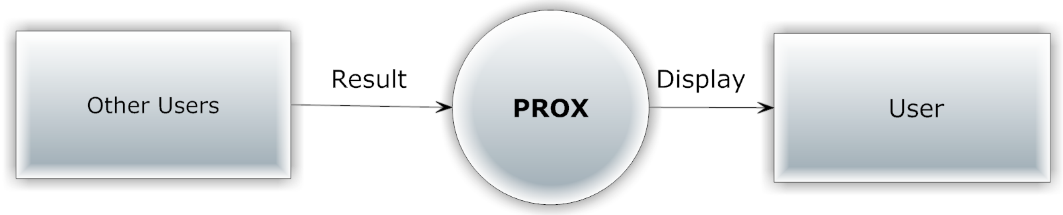


Fig. 5.2: Retrieving Result

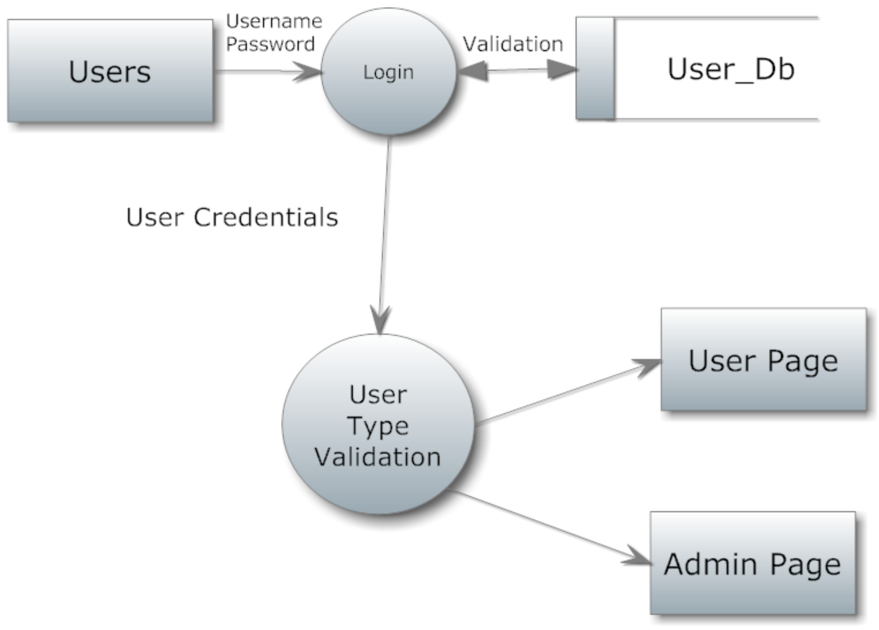


Fig. 5.3: User validation

5.2.3 System Flow Diagrams

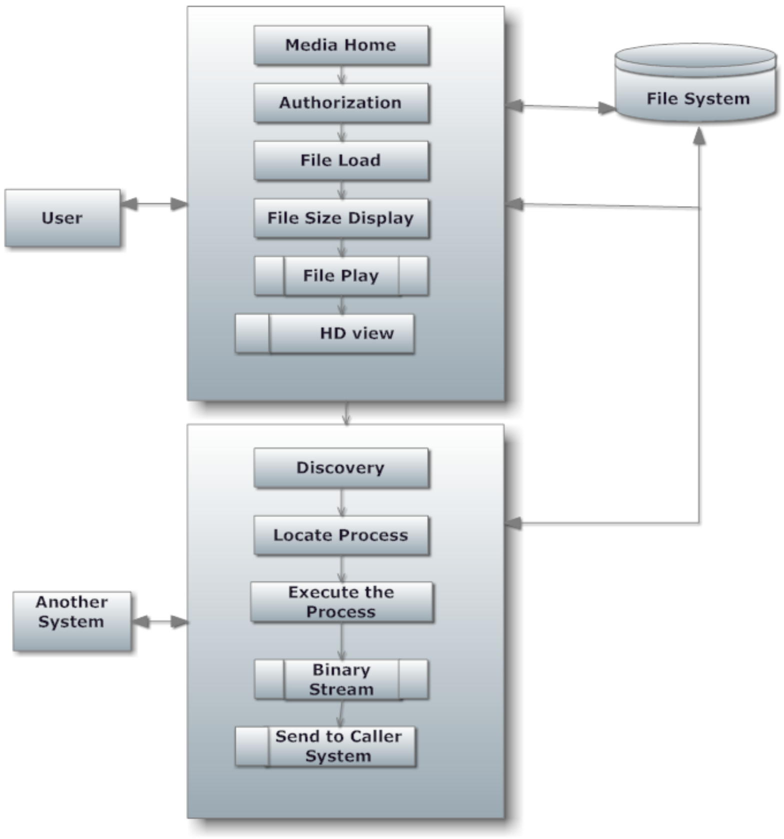


Fig. 5.4: Working of PROX System

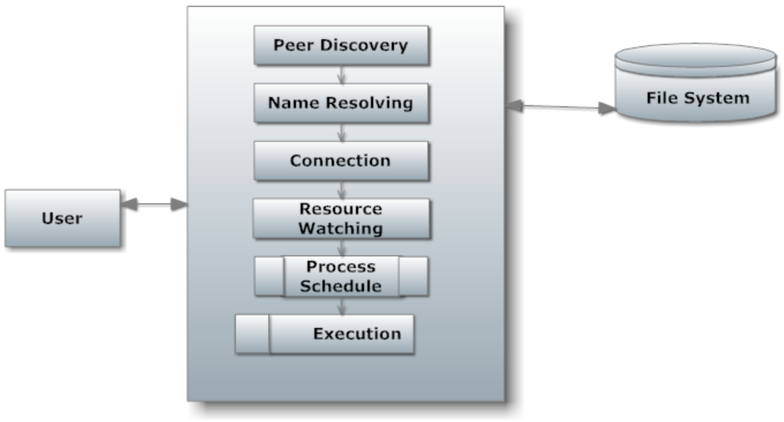


Fig. 5.5: Peer Discovery and Sharing

5.2.4 Sequence Diagrams

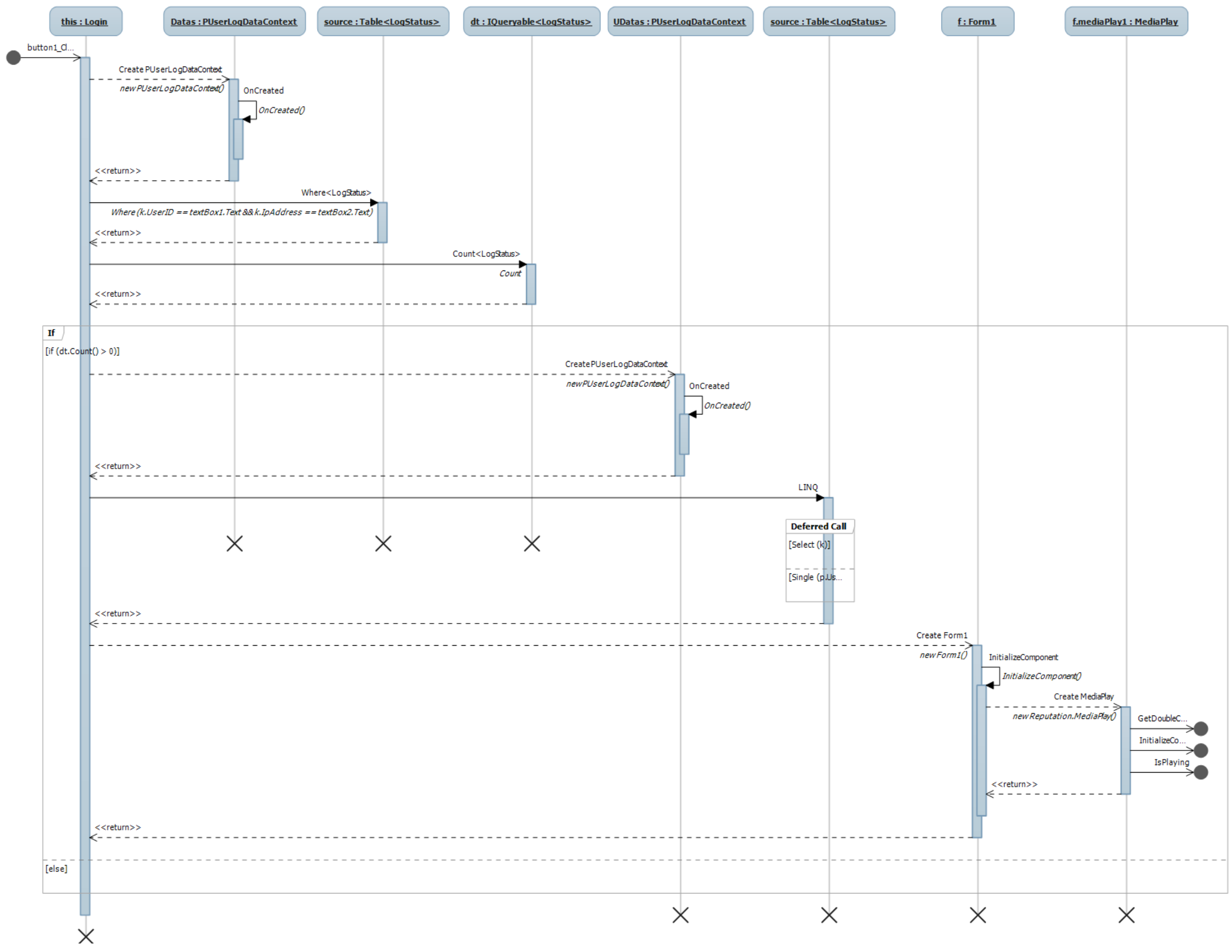


Fig. 5.6: Login function

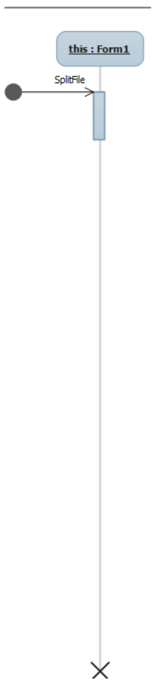


Fig. 5.7: Split File Function

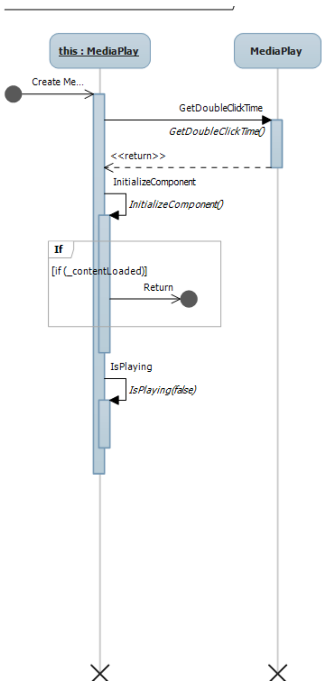


Fig. 5.8: Media Play Function

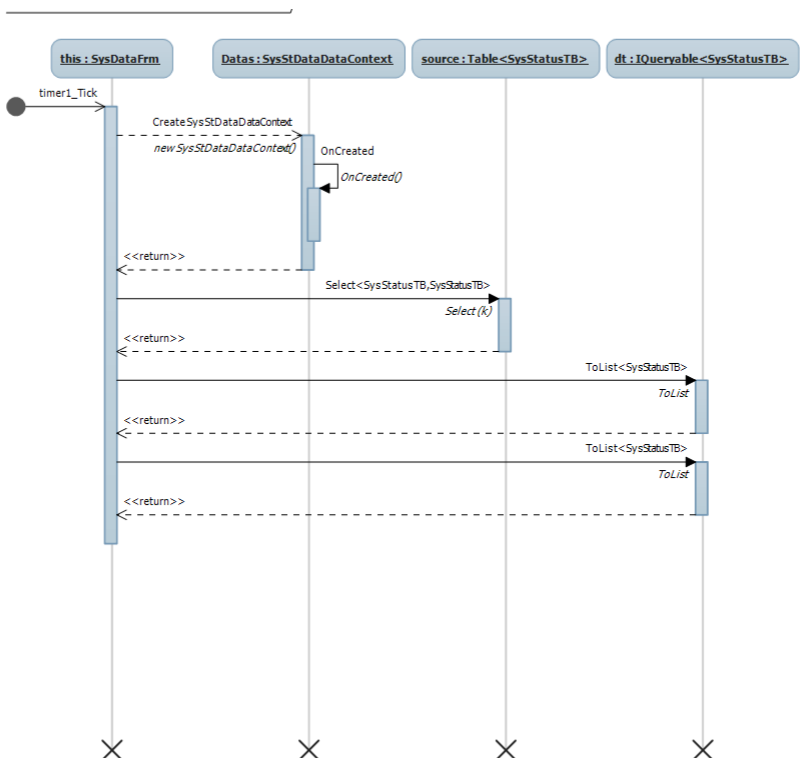

this : SysDataFrm
Datas : SysStDataDataContext
source : Table<SysStatusTB>
dt : IQueryable<SysStatusTB>
timer1_Tick
CreateSysStDataDataContext
new SysStDataDataContext()
OnCreated
OnCreated()
<<return>>
Select<SysStatusTB,SysStatusTB>
Select (k)
<<return>>
ToList<SysStatusTB>
ToList
<<return>>
ToList<SysStatusTB>
ToList
<<return>>

Fig. 5.9: System Data form

5.2.5 Class Diagrams

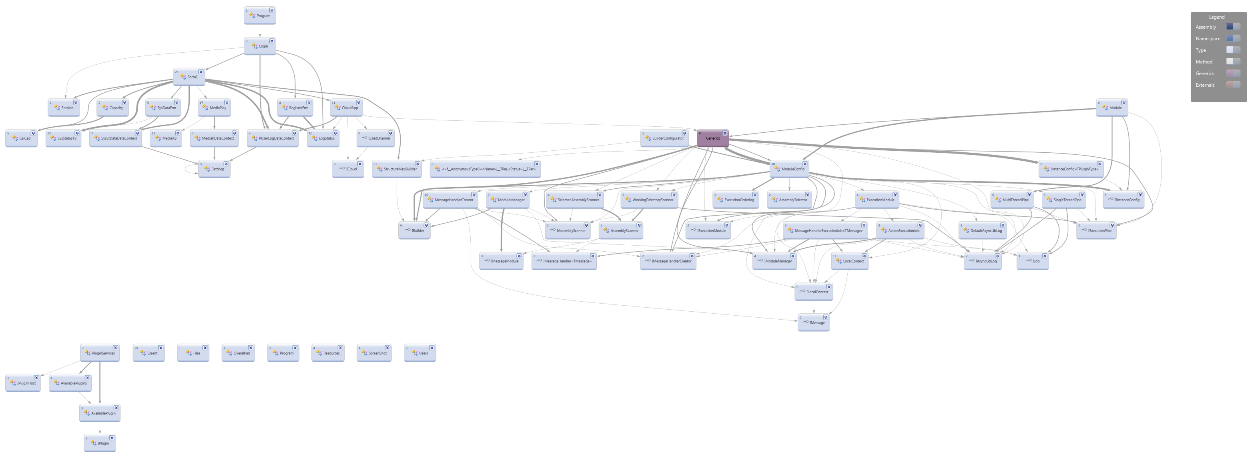


Fig. 5.10: PROX System

CHAPTER 6
IMPLEMENTATION

6.1 Introduction

Basic aim of this project is to create a Peer to Peer content and application streaming media player which increases the efficiency of the video playback for DVD, High Definition Video Content and allow the users to stream the video or music files directly from their computer hard disk to anyone without copying or downloading and enabling them to user their hardware resources.

System implementation is classified into three parts.

- Registration And Cloud Creation

  It deals with creating a cloud among a peer network.Here after automatically identifying the peers using peer name resolver their Peer Name,Peer IP,Peer Country etc are identified.They are stored in a table.Any changes to these such as addition,modification etc are updated automatically.Users can register to the PROX system and can create their accounts.PROX system will help the users to share the hardware resources such as graphics card within a defined network.

- Scheduling

  It deals with the capacity calculation and other operations such as Partition, pre fetching,prioritization.Capacity calculation calculates the capacity of each peer among a network.The calculation uses values such as upload capacity,download capacity,bandwidth speed,total capacity etc.After this, partitioning of processes and threads are done.They are fragmented and prioritized based on threads.Each thread get executed on various systems depending on the reputation based allocation scheme.

- Application Part

Here a Peer to Peer content and application streaming media player which increases the efficiency of the video playback for DVD, High Definition Video Content and allow the users to stream the video or music files directly from their computer hard disk to anyone without copying or downloading and enabling them to user their hardware resources.

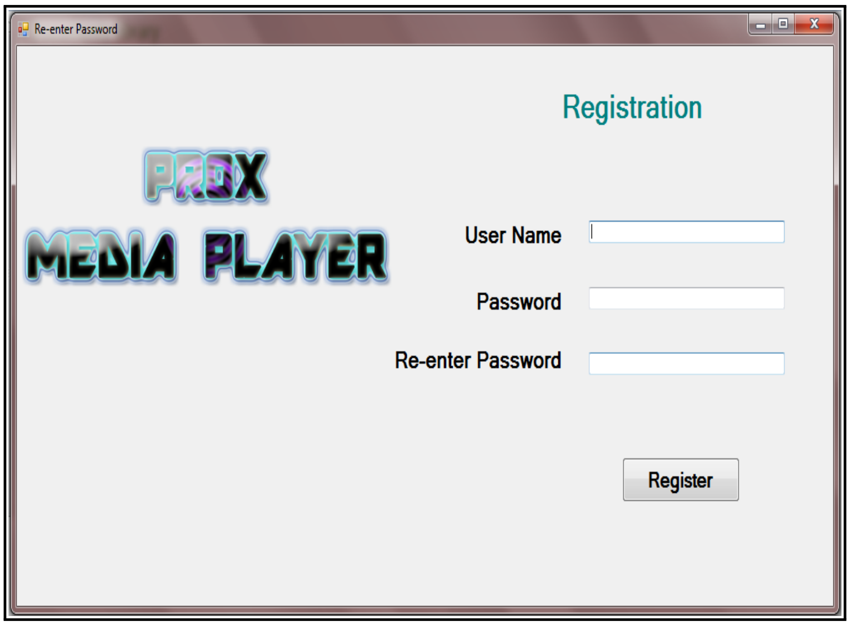


Fig. 6.1: Screenshot 1 Registration

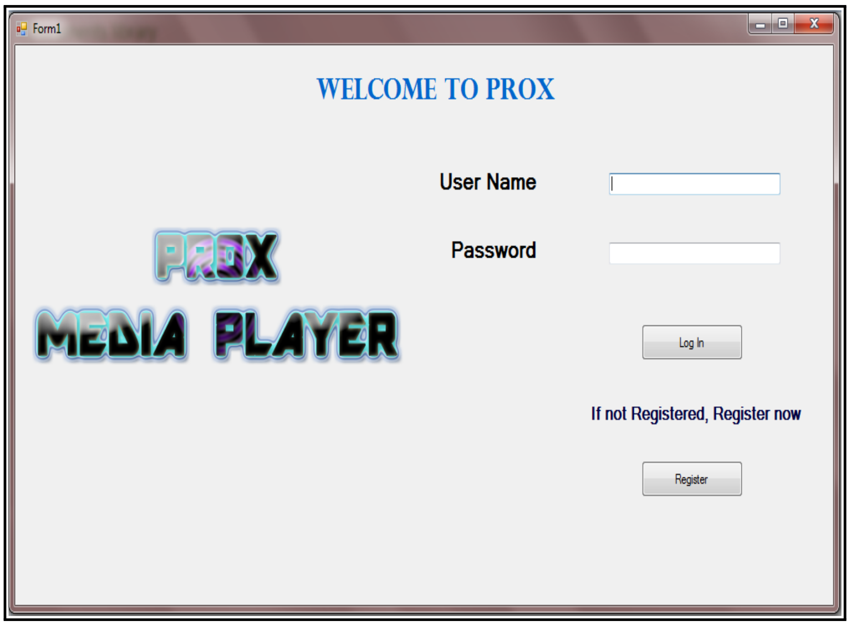


Fig. 6.2: Screenshot 2 Login

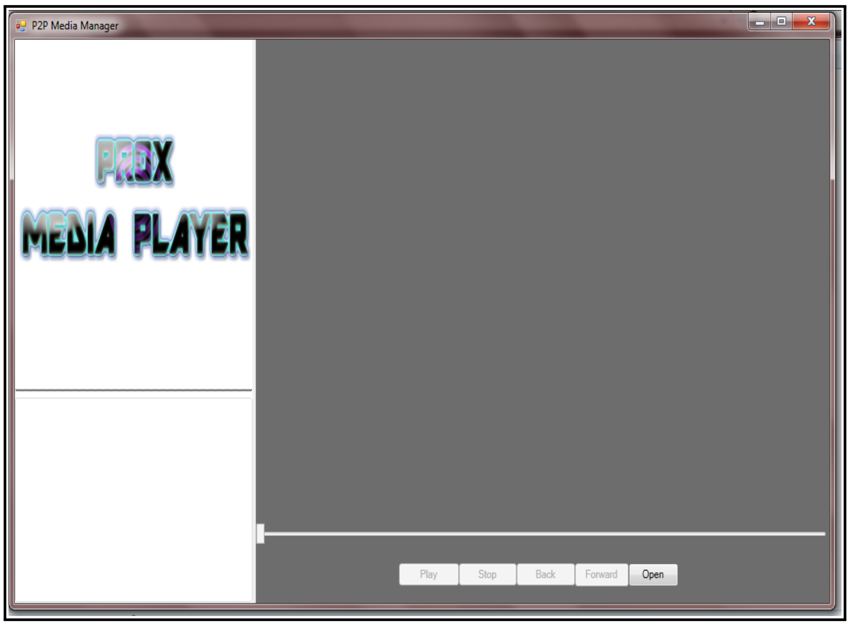
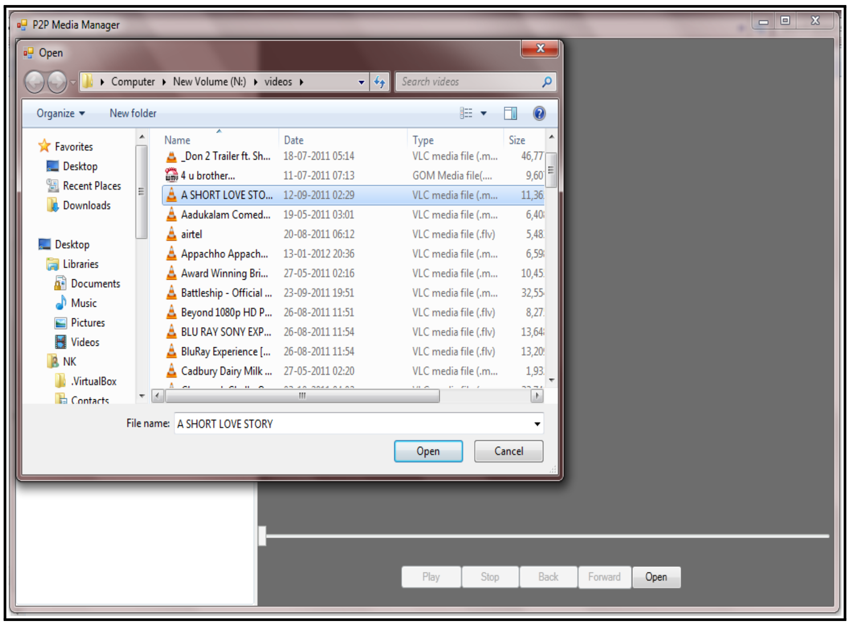


Fig. 6.3: Screenshot 3 Media Player



Name	Date	Type	Size
_Don 2 Trailer ft. Sh...	18-07-2011 05:14	VLC media file (.m...	46,77
4 u brother...	11-07-2011 07:13	GOM Media file(....	9,60
A SHORT LOVE STO...	12-09-2011 02:29	VLC media file (.m...	11,36
Aadukalam Comed...	19-05-2011 03:01	VLC media file (.m...	6,40
airtel	20-08-2011 06:12	VLC media file (.flv)	5,48
Appachho Appach...	13-01-2012 20:36	VLC media file (.m...	6,59
Award Winning Bri...	27-05-2011 02:16	VLC media file (.m...	10,45
Battleship - Official ...	23-09-2011 19:51	VLC media file (.m...	32,55
Beyond 1080p HD P...	26-08-2011 11:51	VLC media file (.flv)	8,27
BLU RAY SONY EXP...	26-08-2011 11:54	VLC media file (.flv)	13,64
BluRay Experience [...	26-08-2011 11:54	VLC media file (.flv)	13,20
Cadbury Dairy Milk ...	27-05-2011 02:20	VLC media file (.m...	1,93

File name: A SHORT LOVE STORY

Fig. 6.4: Screenshot 4 Browsing

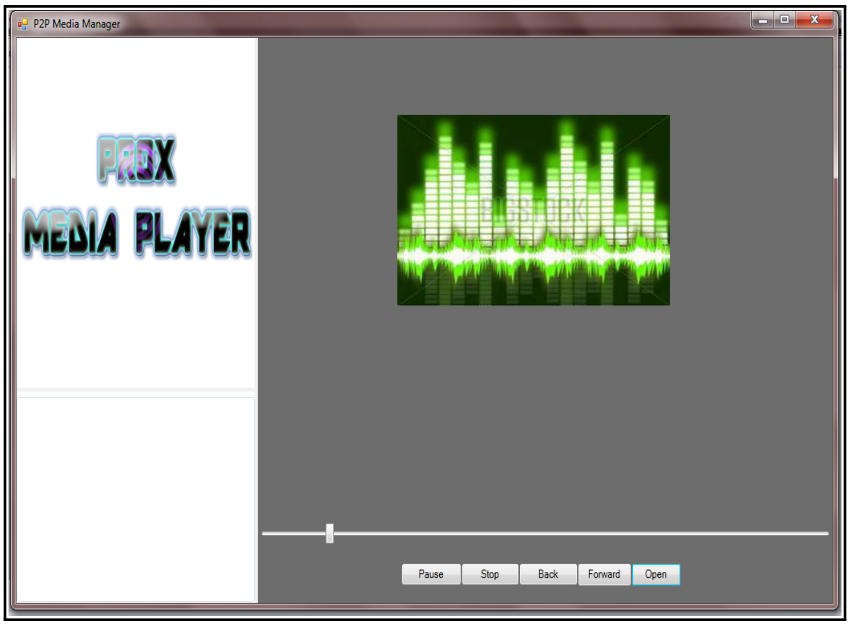


Fig. 6.5: Screenshot 5 Music play

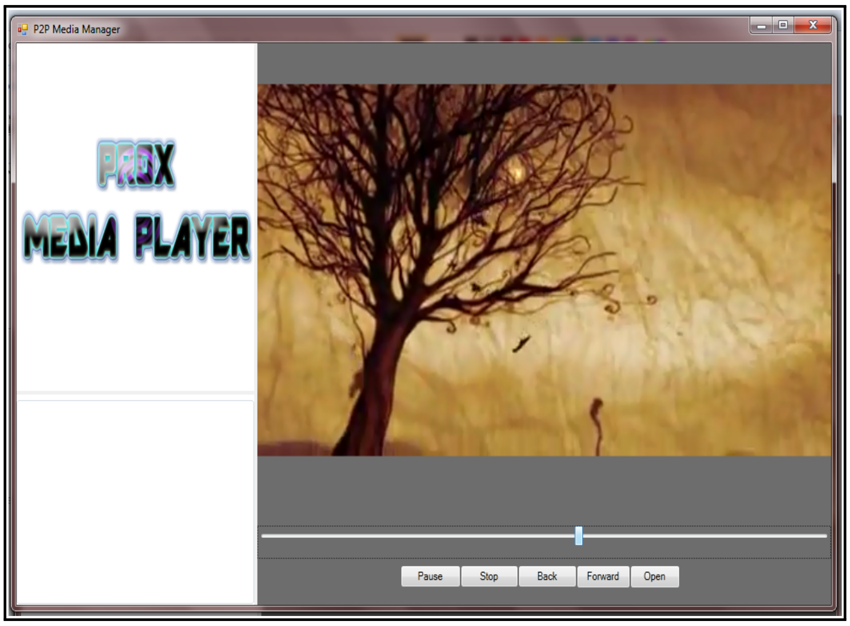


Fig. 6.6: Screenshot 6 VideoPlay

6.2 Limitations

The main limitation of PROX system is that the hardware sharing become impossible if there is no monitor configuration compatibility.

CHAPTER 7
TESTING & MAINTENANCE

7.1 Tests

7.1.1 Unit Testing

This is the first level of testing.Unit testing is done during the coding phases and to test the internal logic of the module.It refers to the verification of a single program module in an isolated environment.After coding each dialogue is tested and run individually.Logical errors found where corrected.

Here the registration part is tested by adding some dummy users.Each user part is checked individually.Capacity calculation using a single user is tested.PROX system using a single user within a network is tested.User details of each individual user such as status of user is tested using various cases.

Media Player is tested using video and music files.Each one is tested by using files having different extensions.Working of each buttons in the panel are tested.The various buttons are play,stop,backward,forward,open.

7.1.2 Integration Testing

When the unit tests satisfactorily conducted,the modules can be gradually integrated and tested.For example,Working of media player is checked after a successful login of a new user.

Splitting and execution of threads in low and high configured systems are tested.Sharing of various hardware resources are checked clearly.

7.1.3 System Testing

The entire application as a whole is tested.Thread execution is tested as a whole.Splitting,fragmentation and prioritization are analyzed.Efficiency of media streaming with and without the PROX system are compared.

7.2 Maintenance

Software maintenance is the modification of a software product after the delivery to correct faults,to improve performance or other attributes.Necessary maintenance are done depending on the PROX system usage.

CHAPTER 8
CONCLUSION

8.1 Introduction

The main objective of this project is to create a Peer to Peer content and application streaming media player which increases the efficiency of the video playback for DVD, High Definition Video Content and allow the users to stream the video or music files directly from their computer hard disk to anyone without copying or downloading and enabling them to user their hardware resources.

Application developed using the current programming platforms executes on a single system, when the application too large.To improve the large scale application performance, data management, accessibility we propose a new kind of programming paradigm through which application utilizes the hardware of nearby system to execute the part of the programmes using direct connection (Peer to Peer).Application execution is scheduled automatically with maintaining the executed program part, executed data part in an stack of task memory on the application system. Results from the other computers are transported back to the application running system results are added to the application.

8.2 Future work

Monitor sharing may be a challenging work which may help to evolve more new technologies.

REFERENCES

[1] A. Zomaya, *”Algorithms and Parallel Computing”*, Series, Ed. Wiely Series.

[2] J.-R. J. Shun-Yun Hu and B.-Y. Chen, “Peer-to-peer 3d streaming,” *National Central University*.

[3] L. Y. Sbastien Vnot, “Peer-to-peer media streaming application survey,” *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*.

[4] M. D. G. A. Jan S., Rellermeyer, “Engineering the cloud from software modules.”

APPENDIX

The important code section is added below.

Media Player

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.IO;
using System.Runtime.InteropServices;
using System.Windows.Controls.Primitives;
using System.Windows.Threading;

namespace Reputation
{
    /// <summary>
    /// Interaction logic for MediaPlay.xaml
    /// </summary>
    public partial class MediaPlay : UserControl
    {
        DispatcherTimer timer;
        public MediaPlay()
        {
            InitializeComponent();
            IsPlaying(false);
            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromMilliseconds(200);
            timer.Tick += new EventHandler(timer_Tick);
            MediaEL.LoadedBehavior = MediaState.Manual;
          }
```

```
private void btnOpen_Click(object sender, RoutedEventArgs e)
{
    Microsoft.Win32.OpenFileDialog dlg = new
        Microsoft.Win32.OpenFileDialog();
    dlg.Filter = "Video Files (*.avi)|*.avi|MP4 Video Files
        (*.mp4)|*.mp4|Video Files (*.wmv)|*.wmv|Music Files
        (*.mp3)|*.mp3";

    dlg.ShowDialog();
    if (dlg.FileName.Contains(".mp3"))
    {
        image1.Visibility = System.Windows.Visibility.Visible;
    }
    else
    {
        image1.Visibility = System.Windows.Visibility.Hidden;

    }

    MediaEL.Source = new Uri(dlg.FileName);
    btnPlay.IsEnabled = true;
}

#region IsPlaying(bool)
private void IsPlaying(bool bValue)
{
    btnStop.IsEnabled = bValue;
    btnMoveBackward.IsEnabled = bValue;
    btnMoveForward.IsEnabled = bValue;
    btnPlay.IsEnabled = bValue;

    seekBar.IsEnabled = bValue;
}
#endregion

#region Play and Pause
private void btnPlay_Click(object sender, RoutedEventArgs e)
{
    IsPlaying(true);
    if (btnPlay.Content.ToString() == "Play")
    {
        MediaEL.Play();
        btnPlay.Content = "Pause";
    }
    else
    {
        MediaEL.Pause();
        btnPlay.Content = "Play";
    }
}
#endregion

#region Stop
private void btnStop_Click(object sender, RoutedEventArgs e)
{
    MediaEL.Stop();
    btnPlay.Content = "Play";
    IsPlaying(false);
    btnPlay.IsEnabled = true;
}
#endregion

#region Back and Forward
private void btnMoveForward_Click(object sender, RoutedEventArgs e)
{
    MediaEL.Position = MediaEL.Position + TimeSpan.FromSeconds(10);
}

private void btnMoveBackward_Click(object sender, RoutedEventArgs e)
{
    MediaEL.Position = MediaEL.Position - TimeSpan.FromSeconds(10);
}
#endregion



  #region Seek Bar
private void MediaEL_MediaOpened(object sender, RoutedEventArgs e)
```

```
{
    if (MediaEL.NaturalDuration.HasTimeSpan)
    {
        TimeSpan ts = MediaEL.NaturalDuration.TimeSpan;
        seekBar.Maximum = ts.TotalSeconds;
        seekBar.SmallChange = 1;
        seekBar.LargeChange = Math.Min(10, ts.Seconds / 10);
    }
    timer.Start();
}

bool isDragging = false;

void timer_Tick(object sender, EventArgs e)
{
    if (!isDragging)
    {
        seekBar.Value = MediaEL.Position.TotalSeconds;
        currentposition = seekBar.Value;
    }
}

private void seekBar_DragStarted(object sender, DragStartedEventArgs e)
{
    isDragging = true;
}

private void seekBar_DragCompleted(object sender,
    DragCompletedEventArgs e)
{
    isDragging = false;
    MediaEL.Position = TimeSpan.FromSeconds(seekBar.Value);
}
#endregion

#region FullScreen
[DllImport("user32.dll")]
static extern uint GetDoubleClickTime();
```

```
System.Timers.Timer timeClick = new
    System.Timers.Timer((int)GetDoubleClickTime())
{
    AutoReset = false
};

bool fullScreen = false;
double currentposition = 0;

private void MediaEL_MouseLeftButtonUp(object sender,
    MouseButtonEventArgs e)
{
    if (!timeClick.Enabled)
    {
        timeClick.Enabled = true;
        return;
    }

    if (timeClick.Enabled)
    {
        if (!fullScreen)
        {
            LayoutRoot.Children.Remove(MediaEL);
            this.Background = new SolidColorBrush(Colors.Black);
            this.Content = MediaEL;

            MediaEL.Position = TimeSpan.FromSeconds(currentposition);
        }
        else
        {
            this.Content = LayoutRoot;
            LayoutRoot.Children.Add(MediaEL);
            this.Background = new SolidColorBrush(Colors.White);

            MediaEL.Position = TimeSpan.FromSeconds(currentposition);
        }
        fullScreen = !fullScreen;
    }
}
#endregion
```

```
        private void ChangeMediaVolume(object sender,
            RoutedPropertyChangedEventArgs<double> e)
        {
            MediaEL.Volume = volumeSlider.Value;
        }
    }


}
```