Scalable
Spatially Aware Media Sharing Display System

By
Patrick Menard
Bachelor of Science, Computer Science and Engineering

Submitted to the
Department of Electrical Engineering and Computer Science

August 16, 2004 [September 2004]

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Author_____
cal Engineering and Computer Science
August, 13, 2004

Certified by
John Maeda
Media Laboratory
Thesis Supervisor

Accepted by
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Scalable
Spatially Aware Media Sharing Display System

By
Patrick Menard
Bachelor of Science, Computer Science and Engineering

Submitted to the
Department of Electrical Engineering and Computer Science

August 16, 2004

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science.

Abstract:

The scalable spatially aware media sharing display system provides an efficient and convenient means of harnessing media messaging in global communications. A three-tiered system of input, control, and output creates a medium for communication and data sharing via varied media types for relevant and enhanced contextual experiences. It is built as a platform independent infrastructure for sharing and/or displaying various media types onto a grid of, or singularly placed, target display nodes while accommodating dynamic growth in its support of on-the-fly display node assimilation. The system promotes interfacing flexibility thus allowing multiple devices, extant or yet to be, to connect and fully exploit its capabilities. In addition, the system supports an architecture that can accommodate loosely coupled parallel tasks thus exhibiting the qualities of a dynamic parallel cluster.

Acknowledgements

I would like to thank God for all that He is and for all that he has done. Without Him, I would not have come this far. I would like to thank my family for all of the support they have given me throughout the years and for *being* family in the true sense of the word. I would like to thank Carlos Rocha, Marc Schwartz, Noah Fields, James Seo, Hilary Karls, Mimi Liu, and Allen Rabinovich for all of their help throughout the research period and during the last push. You all are the best. I would like to thank Heather Pierce for her awesome efforts and support to the group and Fred Lee for taking a last minute read. Finally, I would like to thank John Maeda, an advisor whom words cannot describe.

Table of Contents

1 Introduction

The need to effectively communicate ranks high among the list of factors a sustainable society requires to function and survive. Communication in the form of media has been found to perform this task with great clarity and depth of context. As a result, the population's hunger for accessing, displaying and sharing media with family, colleagues, businesses and co-workers pushes networking, graphics processing, and computer technology to become faster and more reliable. In an effort to satisfy the need of media communication, mechanisms that perform this specific task now permeate our every day lives as evidenced by the extensive variety of and growing market for next generation, media-rich handhelds, cellular devices, and subscription services.

The advent of such pervasive digital content creators, such as digital cameras and smart phones, has also spawned a new problem of information overload. Their great abundance has allowed the casual user to amass large and unwieldy libraries of digital content. In another scope, companies that create digital content as a means for existence face this problem in greater orders of magnitude.

Moreover, advanced or creative individuals or organizations may endeavor to produce custom forms of media as a means of digital expression, information processing, or communication. Such custom media may require notions of state and the ability to execute, like a program. This kind of media also opens the door for mechanisms that can perform a measure of parallel computation on real data sets or the ability to concurrently perform multimedia tasks and computationally intensive jobs.

In addition, communication measures require simplicity of use and configuration in order to be effective. A phone simply works. A television, by the press of a single button, simply works. These are the technologies that add a level of richness to communities without burdening them with added complexity.

Finally, the urgency of being connected and up to the minute motivates the need to improve this form of communication to the point of effectively zeroing the distance between disparate groups.

Therefore it is critical, especially in urgent situations, that mechanisms of media communication instantaneously connect specific groups with minimal effort and boast an efficient means of gathering, processing, and centrally storing input from multiple sources.
Furthermore, a solution would ideally be platform independent, scalable, spatially aware, media independent to the extent possible, and reasonably fast. In addition, this form of communication must be able to coordinate with the various input devices that are currently readily available and be flexible enough to accommodate and interface with those devices yet to be developed. Finally, the interfacing and configuration of such a mechanism must be simple and nearly effortless.

For these reasons, I have a developed a system and an architecture that can operate on diverse media with the goals of simple interfacing and making information readily available, easily comprehensible, and realistically portable. More importantly, it endeavors to save time while performing such tasks. Such a system will ignite a spark that will begin to revolutionize the way people, agencies, and businesses communicate. Furthermore, this application doubles as a reason to attempt building a platform independent architecture for media communication and/or parallel computation accessible by dedicated or general communities in an effort to zero the communication distance gap and create opportunities for use of otherwise inaccessible technologies.

1.1 Methodology Overview

My approach is to build a scalable spatially aware media sharing display system that seeks to close the distance gap, and to provide people a means to communicate via media effortlessly and efficiently. This endeavor involves interleaving current technologies with new technology. Solutions for platform independence and media independence

have partially already been developed through technologies like Java and QuickTime. Furthermore, the solution for closing the distance gap has been realized in the form of the World Wide Web. Thus by combining all three technologies the system already begins to accomplish, in theory, some of its goals.

My solution involves creating a client/server architecture that handles input requests from various sources, processes them, and dispatches them to appropriate destinations. The destinations are the displays that will depict the media received. These displays process received requests and perform only those calculations pertinent to its location. Since displays can compute, a conduit for parallel processing among display nodes for efficient means of data manipulation is realizable thus allowing the system to take full advantage of the displays. By dividing the task of content operation among multiple displays, speedup in processing time can be achieved while approaching or achieving the goals previously stated. These specifications motivate some form of configuration requirement for displays connected to the system including and an ability for the system to realize their existence on its network. In order to maintain the goal of simplicity and near effortless user interfacing, the configuration is as minimal as possible leaving the bulk of discovery automatic. Stressing the ability to accommodate multiple clients on multiple platforms also requires a mechanism for assimilating many potential displays as nodes and making the system scalable.

In addition, handling requests from multiple input sources necessitates a protocol of communication between the system and these sources that adheres to platform independence and tolerance of deviations. In other words, the system should allow users to submit various media of differing complexity and area using a systematic foundation that can support such interaction. This implies a structure of communication similar to that of the Internet yet not as rigorous. In essence, data sent and received need to be sent in packets of bytes enclosed in a header and footer. The central core of the system would then need to read the header and decipher the data in accordance to the header's content. Finally the core would assemble another communication data slice with header, data, and footer pertinent to the destination stations to which they belong. Since the system is in

Java, Java-enabled devices can easily plug in by relying on existing Java mechanisms on top of TCP/IP and need not utilize the low level super controlled protocols.

Another important aspect of the system is stability and reliability. By using a client/server model in my approach, the system can and must handle multiple requests cleanly and properly. It must do so in a manner that isolates multiple requests to promote fault-tolerant behavior. Furthermore, the system must be able to heal itself if any of its components do go down. This healing must occur mainly on two fronts: the core and the destination nodes. These two aspects of the system have the potential to be running non-stop for days. Thus resource allocation must be correctly handled as well as cleanup to ensure smooth operation. Also, the core must be able to detect any disconnections from both input devices and destination nodes and update its state respectively. Destination nodes themselves must be able to detect if the core goes down and cleanly disengage to protect against leaks and allow for reconnection.

1.2 Thesis Structure

The rest of this thesis will contain more in-depth discussion on precedents, the system itself, design and implementation, evaluation, and other particulars of the system. Some of the concepts that will be discussed in or arise from this thesis are Shared Display Spaces, Media Clustering and Messaging, Ubiquitous Multi-interface Access and Control, Auto Discovery and Zero Configuration (Rendezvous), Node Assimilation, Distributed Computation, Sensor Networks, and Remote Administrator Control. The following chapter breakdown will expound on the stated goals of the research:

> *Background* discusses past and recent technologies and precedents related to the endeavor of communication via media and its efficiency and impact both on society and the contents of its discourse as well as shared vehicles of expression.
>
> *Early Experiments* will discuss previous systems I have developed that in whole or in part deal with sharing a space for expression among multiple users, a

mechanism by which to do so, and an interface that encourages simplicity and ease of use.

System Components presents a discussion on the various components of the system providing definitions as a foundation for discourse on the further particulars of the system design and implementation.

Design will present a high level description of the map of the architecture detailing the major tiers and communication protocols as well as mentioning specific data structures used in the design endeavor.

Implementation discusses in depth the specific components and framework of the system, including data structures and algorithms.

Usage will provide a framework for areas where the system has or can be used as well as extensions to the system or parts of the system to accomplish other creative or necessary tasks.

Evaluation provides comparisons to other systems, specific achievements of this system as well as its shortcomings, and a discussion of suggestions and future work to improve the system.

2 Background

This chapter is broken into two sections: Precedents and Related Work. In the first section I will present technologies that have reshaped communication, user interaction, and community interaction and act as precedents for the further enhancement of media as a form of efficient communication. The second section will discuss specific related work that either motivated my own work or seeks similar goals either in whole or in part.

2.1 Precedents

Many precedents exist revolving around media as a means of communication to bring communities together for aesthetic, informational, recreational, or educational needs. The three I will present include television, interactive entertainment, and the World Wide Web. All three have revolutionized the way people in all walks of life, even nations, communicate, interact, and learn.

2.1.1 Television

> *Today we have, in a sense, the transmission of sight for the first time in the world's history. Human genius has now destroyed the impediment of distance in a new respect, and in a manner hitherto unknown. - Herbert Hoover [39]*

Herbert Hoover, then Secretary of Commerce, spoke these words on April 7, 1927 during the first ever demonstration of electromechanical television broadcast in the United States, second to an earlier demonstration by Scotland's John Logie Baird on January 26, 1926 [39,40]. Witnesses of this feat had an idea of how important this achievement in technology was and would become, but there is no way they could have guessed just how immense and pervasive such an achievement truly became. The invention of the television sparked a new era in communication that would forever change how people interact and acquire information, much like the telephone invented four years prior had

done. In fact, the first transmission used telephone lines as its conduit, followed by radio transmission of the broadcast shortly after.

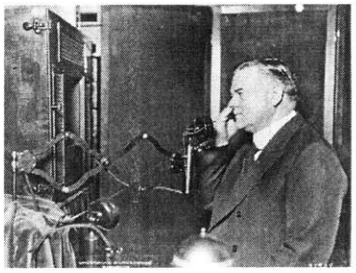


Figure 2.1: Herbert Hoover giving the address [39].

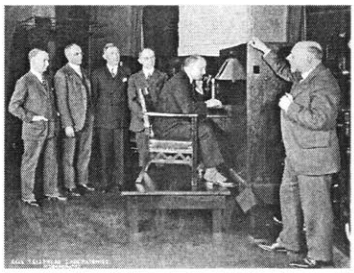


Figure 2.2: AT&T's Walter Gifford observing Hoover via television [39].

Hoover, as well as the scientists and reporters, understood that the distance gap between two communicating parties via visual means had then, at that instant, been virtually eliminated. An article in the New York Times covering the event commented that "time as well as space was eliminated" while expressing how taken by the spectacle witnesses must have been in not only hearing the words forthcoming, but seeing the changes in Hoover's facial expressions -- at virtually the same instant the very person next to Hoover heard and saw his words and expressions [15]. The mere ability of sight added valuable context to the speech being delivered thereby conveying significantly more information than hearing alone could have done.

Shortly after the first transmission, Philo Taylor Farnsworth developed fully electronic television eliminating the need for mechanical intervention [38]. This great feat was followed by color television, among other things, and the first transcontinental broadcast on September 4, 1951 of President Harry Truman's address to the United Nations [39]. Witnesses and viewers as far as England could tune into the broadcast being sent from San Francisco. As time passed, the technology only became better, more accessible, and exceedingly adept at depicting visual information in greater degrees of definition and also leaping over the problem of distance for world communication. Specifically, the advent of satellite television broadcast allowed audiences in England and France to see a press conference conducted by President Kennedy and those in the States to witness "French

singer Yves Montand and the changing of the guard at England's Buckingham Palace" live [39].

In other words, communication via media had become essential to the task of affording people instant access to important events occurring in the now as well as allowing the sharing of cultural talent and/or way of life, live and in living color. Such a scope enhances the experience and information gathering many fold. One would be hard pressed in this day an age to find a home without some form of television present. The medium brings families together for various activities such as major sports coverage, world premieres, home movies, and learning (from sources like the Learning and Discovery channels). Media is so designed as to make even the dullest subject interesting.

Television has also allowed people, at least here in America, the ability to experience in some form events they would not otherwise have seen or experienced. Televised war emerged with the Gulf War in 1991. With Operation Iraqi Freedom in 2003, the advent of 24-hour war coverage entered our society. People at home could now follow the war as it unfolded, literally at the instant events occurred out on the field – something never possible before [26]. This resulted in people being more knowledgeable about current events and the impact actions in government, various agencies, and people were having on the world. In addition, it afforded more people access to such information and the reality of war staring them right in the face as best the News stations could possibly deliver.

Since the advent of this medium of media communication, efforts to amplify that form of communication immediately became necessary. Aiding in the current success of this platform is the fact that, plainly, television works. One need only turn it on, blowing the floodgates open and allowing a rush of communication and information to pour through. Simplicity.

2.1.2 Interactive Entertainment

Interactive entertainment indirectly began in 1952 when A.S. Douglas, as a mechanism to demonstrate his thesis on Human Computer Interaction, constructed a graphical version of Tic-Tac-Toe. At MIT, Steve Russell and a group of students further pushed the idea of video games in 1961 by developing Spacewar, a game that set two players against each other [23]. It was with Spacewar that this new medium of interaction began to be noticed. Much later, arcades were being developed and placed in numerous locations including bars, restaurants, and of course arcades. As further advances in technology emerged, consoles that attached to the television and used it for gaming began to surface. The industry really took flight when Nintendo and Shigeru Miyamoto emerged on the scene [23]. When the Nintendo Entertainment System (Famicom in Japan) was released in 1985 for US markets, a gaming culture movement began to sweep the nation and many other areas of the world.

(a) (b) (c)

Figure 2.3: (a) Nintendo's Super Mario Brothers [42]. (b) John Carmack's DOOM [27] (c) John Carmack's DOOM 3 and his new technology pushing lighting effects multiplying in game realism [7].

Since interactive entertainment began to emerge as a hotbed, its consumers demanded and expected enhanced control, enhanced graphics, enhanced physics, and enhanced AI with every new game released. Such a market, in addition to pushing advances in technology, continually pushes advances in graphics and the display of such media beyond any apparent limits. Consumers wanted more realism both in interaction and experience. This is evidenced in the past by John Carmack's DOOM and in recent history by Rare's Donkey Kong Country and John Carmack's DOOM 3 [23,28]. DOOM

revolutionized gaming by introducing 3-Dimensional environments in first person style on machines barely capable of such processing. The significance of this feat expresses itself by immersing the user in more realistic environments so much that turning a corner could cause a person to jump out of his or her seat in fear – for a second.

Soon other industries, even the government, took notice of the potential of the medium as a training mechanism or its technology as a means for immersive exploration. The US government commissioned Ed Rottberg, creator of Battlezone, to develop an enhanced version of the game for the purposes of military training. The military also used John Carmack's multiplayer DOOM [29,28]. The graphical engines of such games became so advanced that architects began using them to explore designs before embarking on their actual physical creation [28].

Besides use as military training, interactive entertainment is also used in some forms of education as well as professional training:

> *We see games as enhancing the capabilities of gifted teachers, not displacing them with impersonal machines. Yet, games do offer teachers enormous resources they can use to make their subject matter come alive for their students, motivating learning, offering rich and compelling problems, modeling the scientific process and the engineering context and enabling more sophisticated assessment mechanisms.*
>
> *- MIT Professor Henry Jenkins et al [16]*

The capability of interactive entertainment to turn dull content into exciting experiences or merely enhancing an instructor's ability to convey information resides in its ability to immerse a user within the game world captivated by the visuals, the gameplay, the fun factor, and its competitiveness. Companies such as Walmart have commissioned developers to create specific interactive games to teach management how to run a center. Such games are targeted at older audiences in an effort to convey concepts and skills of management, public policy, and health [9,11].

In addition to emerging as an excellent medium for conveying information, the very worlds exist as places for shared interaction. Massively Multiplayer Online games leverage the connectivity of the network to bring together people from all over the world into one massive shared realm. In this world, game state is persistent and thus approaches the reality of scarce resources in the real world. People engaging in these types of games create vast new societies, orders, and laws of existence. Such shared spaces have the power to bring people together in constructive ways enhancing their quality of life and network of friends. Other games, such as Dance Dance Revolution bring on masses of people together in the real world to try their hand at outperforming any dancer who dares step up to the plate or to simply view the spectacle of competition, much like a battle does in the break dancing arena.

For this medium to succeed beyond visuals and the ability to immerse users, the method of control had to be such that anyone can partake in the experience. Like the Television, creators of this medium designed controllers with the minimum amount of buttons necessary that still allowed immense and yet to be imagined control within a gaming environment. The user interface had to be simple if the medium hoped to reach the masses.

2.1.3 World Wide Web

The World Wide Web, an invention developed to enhance the capabilities of the Internet, spawned a vehicle of communication changing all facets of society. The Internet itself was born out of the military's need for and necessity to share information and new technologies readily, easily, and securely to many users at once and quickly [3]. The Advanced Research Projects Agency Network (ARPANET), as it was called went online in 1969. It was based on a packet switching network that allowed a system to conduct communication with more than one machine [3]. Without getting into the specifics of the protocol, a designer, Paul Baran, submitted several proposals fleshing out the system and its behavior, specifically the use of a decentralized network that allowed for multiple paths between machines or contact points and dissecting of network messages into

packets (which he called message blocks) [3,34]. Such a system would eliminate single points of failure and allow the network to possibly heal itself or protect itself from heavy attack. Since messages were broken into packets, it would be possible to reconstruct the messages since entire messages would not be lost in the event of dropped packets. Redundancy was a key aspect of this design for reliability and sustainability [32].

Robert E. Khan, commissioned with the task of somehow connecting the various protocols that emerged from ARPANET within ARPANET, later redesigned the network moving the responsibility of reliability from the network itself to the hosts. Khan and Vint Cerf were able to strip the network down to a bare minimum thus allowing connections between any two networks without requiring knowledge of their sub-networks [22]. This allowed for greater breadth in information sharing.

Since the early internet was a government funded project, its use was restricted for purely non-commercial purposes, i.e. sharing of research between various universities, agencies and companies aiding in the further development of the net or otherwise important technologies. Thus the focus of the Internet remained as a mechanism of communication for the sharing of vital information and the increase in knowledge of all included parties. The other goal of the system at this point and previously mentioned was reliable communication and information sharing to multiple parties at once, quickly and efficiently.

The World Wide Web emerged in 1989 as Tim Berners-Lee envisioned a scheme that would make document location and viewing easier. This vision, again, grew out of a necessity for him and fellow physicists to share relevant information about their respective research endeavors. His project caused the development of the uniform resource locator, the hypertext markup language, and the hypertext transport protocol (URL, HTML, and HTTP respectively). The genius of this invention lay in its simplicity of use. A document anywhere on the web could be located with one line of text. Furthermore, it was backwards compatible with its protocol predecessors. HTML itself

allowed documents to link to yet other documents quickly multiplying the usefulness of the web as a communication mechanism [22].

Despite these feats, the World Wide Web was still lacking, as all information was purely textual. Eventually people figured out that HTML could also be used to embed media such as graphics. From that point on, both the simplicity of its design and focus as an information-deploying medium coupled with ease of document creation and linking caused the web to grow exponentially.

Two recent important instances tested the web's stability as well as its viability as a communications platform. The first was the dot-com bust of 2002 that caused may independent service providers and telecommunications carriers to bottom out. Despite the losses of these carriers, the Internet itself remained intact successfully demonstrating its level of fault tolerance and redundancy [25]. The second incident involved that fateful day, the events of 9-11. On that day, many people frantically turned on televisions and radios. Many others flocked to the web in order to find more information on the state of affairs, some with even more urgency than others. Search engines, traffic directors if you will, were used then to point people to the most up to date information. After all, these mechanisms "know" more of the web than can any single person. A staggering 6000 users per minute anxiously pegged Google's search engine just to find CNN. How many more per minute sought other queries? This sudden need for information proved the medium a necessary mechanism in the eyes of the people for information (as well as Google being viewed as a trusted source to return the most relevant links possible). The event actually triggered a change in Google's strategy of being search king in ways that would further facilitate the discovery of relevant sites and information regarding breaking news. This incident also tested the ability of the Internet to re-route data amidst broken or downed links due to network overload [35].

Figure 2.4: Wikipedia screencap

Today, many technologies (such as Flash, Shockwave, Streaming Media, and Java Script) have been created in order to enhance multimedia expression via the web. Not only has it grown as an important medium to research topics of interest and a source for news, it is also used for many as a source for connecting to peers, friends, family, and the world as well as a place to conduct business. The emergence of numerous weblogs and wikis attest to the inherent desire of people to collaborate on shared spaces and provide graphic media to tell a story or convey the context of an experience. Weblogs are a form of online journal that can be personal or collaborative, even topical, and allow users to post content in an effort to communicate and keep people informed and up to date. Wikis are similar to blogs in regards to collaboration, however they endeavor to provide specific "peer reviewed" content. Wikipedia, a free online encyclopedia, is a prime example where people the world over can add relevant content to grow the encyclopedia while others check the validity of the content in effect creating an ultimate form of shared information with contributors from all walks of life, positions, authority, and renown.

Every piece of technology discussed above with regard to this medium, even the medium itself, grew out of a need to easily share information with many people, quickly and efficiently, and with guarantees of stability, robustness, and sustainability. At the same time, it was pushed to new heights with the need to use media as an effective vehicle of expression.

2.2 Related Work

In recent development, many projects either from industry or academia that revolve around media messaging, shared displays, and both local or distributed information processing and sharing have surfaced. The following projects or products provide a basis and motivation for my work. Each touches on an aspect of my system and seeks to accomplish the specific goal of simplifying or pushing media communication.

2.2.1 Macromedia

Macromedia developed Macromedia Flash Communication Server MX as a tool for enhanced communication between two sources. This solution allows for the use of text, graphics, audio, and video during communication. Furthermore, the solution uses current technology (such as the web) as a point from which to build their technology. The solution involves web interfaces for the parties communicating using Macromedia Flash, the Communication Server to handle the processing of text, graphics, video, and audio, and a database for storing data to allow sharing and a distributed structure. For networking, Macromedia decided on using Real-Time Messaging Protocol (RTMP) whose purpose is efficient message passing and synchronous feeds among multiple clients. In order to form a sense of media independency, Macromedia used a video codec developed by Sorenson Media and an advanced audio codec. It appears Macromedia also aimed to make integration easy with other technologies making the technology more powerful as a tool capable of being used anywhere. Macromedia's packaged solution concentrates more on the ability to send live video synchronously to one or multiple recipients to aid in communication between the parties performing a task and zeroing the distance gap. By building upon their existing Macromedia technology, the system has the added benefit of ease of implementation. People already know how to use Macromedia. Now they can use it as a basis for communication in conducting business and sharing data [17].

2.2.2 UCP Morgen

UCP Morgen has developed a media messaging solution for mobile phone devices that allows messaging with photo, audio, and multimedia. Users can compose messages that incorporate all three into a single message and send it to intended targets. The solution also allows users to compose these messages on a website and send them to mobile units via the website. UCP Morgen's technology is a feature rich solution allowing users to perform other tasks such as sound composing. The solution also involves storage of media messages to some central location accessible by the web and attempts to interface with existing technologies in order to expand its potential capabilities [4]. The system, however, appears to be E-mail taken to the next level. It functions as a method of communication whose primary target is mobile-to-mobile communication. Like E-mail, this is a personalized system.

2.2.3 InFocus LiteShow

InFocus LiteShow is a new system enabling digital projectors to be accessed via wireless protocol (IEEE 802.11b or Wi-Fi). The system allows present connectable devices running the LiteShow mechanism to automatically discover InFocus projectors. Such a system can speed up and change the concept of presentations especially concerning conferences and institutions of learning. Laptops have essentially become mobile devices especially in the communities previously mentioned. When these people are gathered together, the LiteShow technology can help these groups avoid the bottleneck of physically hooking up devices to a projector for the purpose of presenting projects, design concepts, and proposals. The solution also employs a notion of users and authentication giving projectors access control limitations [8]. In a nutshell, this technology creates an environment for communication among multiple users located in a central location using a shared display device.

2.2.4 Stanford Graphics Lab's Multi-Graphics

Stanford Graphics Lab's Multi-graphics project seeks to create a scalable graphics system for clusters. It sought to use parallel computing to speed up graphics rendering for tiled displays and/or interactive murals. In order to perform the task of separating computation at a low level, this solution is closely linked to hardware. By harnessing the graphics card, the process of rendering and computation can be sped up by assigning specific graphics cards tasks related to the display device to which it is attached. In addition, the system aims to accept input from multiple applications issuing graphics streams. In order to handle such communication, the system requires that each stream insert synchronization commands (since each stream uses a different graphics context). The system then receives these streams, reorders them, processes them, and produces the desired output [21]. WireGL, an extension to this project, seeks to ensure shared usability of scalable displays by unmodified existing applications. Doing so opens the door for simple interfacing by many potential user groups. WireGL accomplishes this by standing in for a system's OpenGL driver. Thus, the function of translating graphics to a scalable display is transparent to an application [24].

2.2.5 POOCH

Dauger Research, Inc.'s Parallel OperatiOn and Control Heuristic application (POOCH) is a system that networks Macintosh computers together to unlock the power of parallel computing resulting in high performance processing. Its primary goal is to simplify the task of initiating a parallel computing context and facilitating its construction using one, two, or multiple nodes. Part of the implementation uses Rendezvous networking, a technology allowing auto discovery of nodes connected to a local network. Jobs are distributed across nodes as they hop onto the network reducing the time required to perform the total computation for a given problem. NASA and other organizations and institutions of research are currently using POOCH's abilities for its simplicity and usability. POOCH also utilizes the Message Passing Interface (MPI) [12].

2.2.6 Xgrid

A very recent development from Apple and first announced on January 7, 2004 with preliminary documentation on March 17, 2004, Xgrid endeavors to provide a simple mechanism to harness the power of parallel computation among Macs on a network [44]. In that respect it is similar to POOCH. Utilizing Apple's zero configuration scheme, Rendezvous, Xgrid is able to create spontaneous clusters to perform arbitrary computations [10]. It is composed of three parts: a client, a controller, and an agent. The client functions as the request mechanism, the controller as the dispatcher, and the agent as the computation node. Xgrid is designed for batch or workload processing [45]. In other words, the more embarrassingly parallel the problem, the better. Since it is not dedicated to specific tasks, it in theory can support any kind of embarrassingly parallel type of computation [10]. It is also not limited to locally networked computers (which harnesses the power of Rendezvous and zero configuration) but can accept IP specific destinations to access machines in the wider web [45]. This technology is still under development and is currently more suited to scientists and researchers performing computation on large datasets.

2.2.7 Extreme Thumbnail Generator

This program gives a user the ability to create thumbnails of digital libraries residing locally on user machines. It also allows users to create potentially stunning online thumbnail galleries. This glorified GUI supplies the casual user with engaging and visually appealing customizable templates. The interface focuses on simplicity to eliminate any need for programming knowledge. It supports the common image formats (jpg, gif, bmp, tiff, etc.) and also provides some image editing.

The application boasts numerous possible functions aimed specifically at information portability. It can be used for a variety of applications like online catalogs, goods and services presentations, picture galleries, family and wedding albums and even marketing materials. This application costs $34.95 and can be used as trialware for 20 days. For

handling jobs locally and abstracting away complexity for users wishing to create web pages and a variety of other utilities, this application performs the task. Unfortunately it is not free and only operates on the Windows platform. It is also not designed, so it seems, as a tool for simply churning out thumbnails as a unique goal. As a result, for exceptionally large tasks, this generator may not be optimal.

2.2.8 Thumbnail Generator

Developed as a script that uses the Image::Magick module, the Perl version of a product with this name allows a user to create thumbnails of local images with specified conditions such as aspect ratio, custom sizing, filename augmentation, and local storage. This application's aim is to churn out thumbnails at the behest of a user. Since it is in Perl, it functions as a command line interface and likely fancies programmer type individuals. However, being command line operated, the application achieves extreme simplicity and extensibility for scripting applications. By lacking aesthetic quality, the thumbnail generator is very small with regard to file size. However, it does require users to have the optional Perl package Image::Magick. This software is free and runs on Linux like platforms. As a Perl script, the program lends itself more to churning out thumbnails in large quantities. Multiple files can be processed in one call by using the Linux wildcard operator ("*") however it is not clear whether or not the program can operate recursively on directories. Again, this program operates serially on a given input.

Similar to the Perl thumbnail generator, the C# application has the ability to generate thumbnails from a large input of various images. In addition, this application can convert a large body of images from one file format to another and support multiple languages. The application boasts a very simple GUI interface again attempting at abstracting away complexity. It is unclear whether or not this program can select directories or recursively generate thumbnails among directories. This software is freely available as well as its source. Unfortunately, the generator also only operates on the Windows platform. The design seems to be focused a little more on interaction than pure compression (though much less than the Extreme Thumbnail Generator). The

performance on massive digital libraries, as a result, may not be as optimal. This software does not support any remote uploading or downloading.

2.2.9 Ximage Thumbnail Generator

This is an ASP.NET script that takes as input a URL, along with desired width and height dimensions, and outputs a thumbnail. The form of output is uncertain since the developer's site seems inaccessible. For this reason, it is also unclear about the processing ability of this software. Since it takes a single URL, it is likely to perform tasks one image at a time. On the other hand, if it employs crawling, it may be able to take in a large input. The free software operates on both Windows and Unix platforms. Unfortunately, it seems to be non-existent.

2.2.10 IrFanView

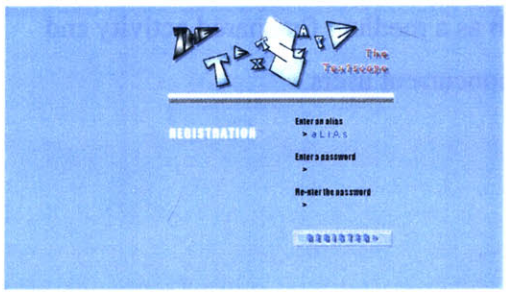
IrfanView is a fast, small, and compact graphic viewer for Windows 9x/ME/NT/2000/XP. It supports a thumbnail/preview option and batch image conversion. It is not clear whether the thumbnail/preview option is thumbnail "generation" or merely a thumbnail "preview". If generation, it is not clear in what quantities or what scale. The killer aspect of this application is its vast support for multiple image, audio, and video formats. The closest to an all in one application, this software exhibits media independence and freedom of creativity. For designers, this is a great tool requiring hands-on interaction. For massive jobs, this software will not be ideal. It is free for non-commercial use and runs only on the Windows platform.

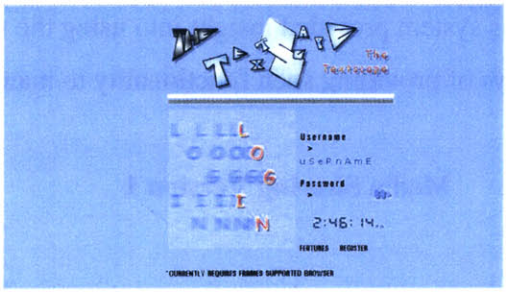
3 Early Experiments

In this chapter I will present earlier experiments that shared, to some degree, the goals or components of the current system. The Textscape was a project aimed for deployment on the web as a multi-user shared space for story telling, world constructing, and community building. The Media Sharing project was a precursor to the developments of this thesis testing the viability of a Media Sharing framework in a local area context. OpenGL Sharing is an extension to the Media Sharing framework aimed at giving a user precise control over interactive elements across multiple computers.

3.1 The Textscape

The Textscape was initially conceived as an interactive outlet for creative textual expression in the emerging OpenAtelier Global Studio birthed within the Aesthetics + Computation Group and continuing in the now renamed Physical Language Workshop. It sought to connect people from all over the world in a way that sparked imaginative thought while maintaining simplicity. Modeled after Multi-user Dungeons (MUDs), The Textscape would run on the web itself via python and cgi technologies and providing the illusion of state. The game world, as well as user state, would reside on the server. The two main reasons for keeping everything on the server were first to force development to be as lightweight as possible stripping away any unnecessary code, and second so that the system simply worked – like television. No further installation or instructions would be required other than to point ones browser at the site, and type a natural language command into the user input field.





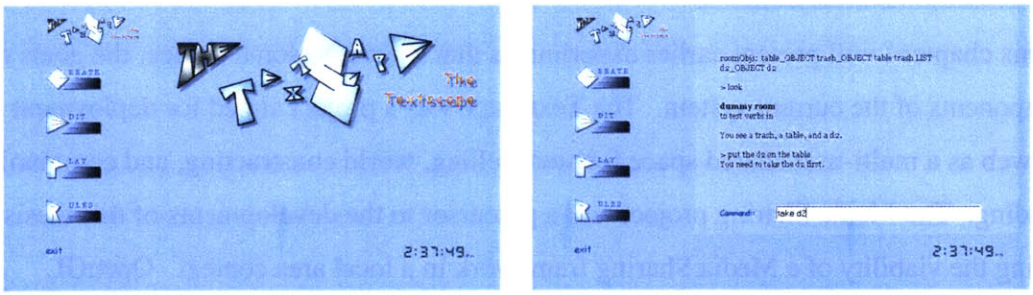


Figure 3.1: Screen grabs of The Textscape including registration, login, in game, and welcome screens (clockwise starting from upper left).

The difference between this system and the already established MUD was its focus on world creation by the community itself. Not entirely game and not entirely chat, this hybrid sought to merge the two in the hopes of creating a community sustaining and persistent realm. Users were given the ability to create items loosely described as objects and rooms. With enough creativity, a user would be able to create any kind of object. Rooms themselves could be s rooms in the natural sense of the word, hallways, or vast open spaces. Furthermore, any user could interact with these objects and locations, create rules for interaction, and develop story lines for interactive use by yet other users.

Since the system mechanism allowed for collaboration, there was a vision that constantly growing and intertwining game environments would ensue. In addition, other users could edit created objects and rooms once the primary creator let the object loose in the world. Designed for simplicity, navigating the world took the form of a single input text field and submit button. A pseudo-natural language engine was written for the system to facilitate transparent user interfacing.

This system provided insight into using the web as a medium for shared activity and ways of providing such functionality to many concurrent users.

3.2 Media Sharing Version 1

The first incarnation of the Media Sharing System acted as a mechanism for local area systems to share images and video quickly and easily. It is based on the Rendezvous technology for automatic discovery of potential client machines. Rendezvous is a protocol designed by Apple that allows computers on the same local network to discover services being published by other computers transparently to a user. By doing so, no in-depth knowledge of networking is required on the part of the user running an application utilizing Rendezvous. Instead, the application simply works as if by magic.

Because of Rendezvous, this system was limited to sharing media with computers locally present on the network. However, simplicity of utilization was supreme. Users could drag and drop images on their local machines into a sharing window. If the user wished to share the media with people on the network, he or she would then click a button that would initialize that process. Computers on the network would resolve this service and immediately load the image automatically.

Once this form of sharing was established, the system was augmented to allow the sharing of movies in addition to images while using the same display mechanism for both. In order for this to happen, the display end of the system was augmented to use QuickTime in the view window due to its support for multiple media formats.

The next step in the development called for scaling the shared media across computers structured as a grid. This endeavor would bring about the notion of a shared display space. The goal was to take an image and, by scaling across multiple screens, elicit a new kind of user experience from the reality of the image's immensity.

Figure 3.2: The iControlCenter's controller interface for the iNode array.

Figure 3.3: Screen shot of a full screen output client display of an iNode station.

To accomplish this feat, the two parts of the system became a control center and an iNode display client. The Rendezvous scheme itself was mutated in a way to change the model from the control center publishing as a sharing service to the nodes themselves publishing as display services. In this way, the control center converted to a pointing device rather than a pointed to device. Zero configuration still surrounded the system, but by allowing the nodes themselves to publish under different names depending on the context, the control center gained a measure of targeting control over which nodes received media, counter to the free for all architecture of the previous incarnation.

In addition, the design endeavored to remove the need for the control center to know the exact configuration of the grid. All it would need is the knowledge that the computer existed on the network. The nodes themselves kept track of their position in the grid thus exhibiting spatial awareness. When a user at the control center end decided to share media across all of the nodes, the nodes themselves would interpret how to go about breaking up the media. A user could easily scale media across the whole array, or send it to be individually displayed. The ability to scale a movie across the screen had not been implemented.

The problem with completely isolating the control center from spatial knowledge surfaced when the idea of sharing the media to different configurations within the same grid emerged. Since the control center had no knowledge, it could not give some form of

direction or hint to the nodes themselves. This would then require the nodes themselves to adopt a more complicated scheme of spatial awareness. Other problems with the system were limitations on number of times media could be shared. The underlying scheme of publishing, discovery, and resolving mandated by Rendezvous apparently had limits on the number of times a resolve could occur. Once that threshold was reached, no more media could be shared until the control center itself was restarted. The application was also limited in its scope. Since it was being developed as a scheme for sharing media locally, this was not much of a problem only a result of a design decision.

This system succeeded, however, in hiding the complexity inherent in the networking beneath the hood from the user. Instead, a user could concentrate on how exactly to make best use of the new visual display medium rather than figure out how it works. One could, among other things, create some intense pattern of sound, imagery, and video to overload the senses of an audience thus creating a masterpiece.

3.3 OpenGL Sharing

The OpenGL Sharing system utilized Media Sharing technology to demonstrate the possibility of explicit user control over more dynamic data in a local area context across multiple client machines. The development of this system also motivated the design of an improved grid structure software representation for the spatially aware context.

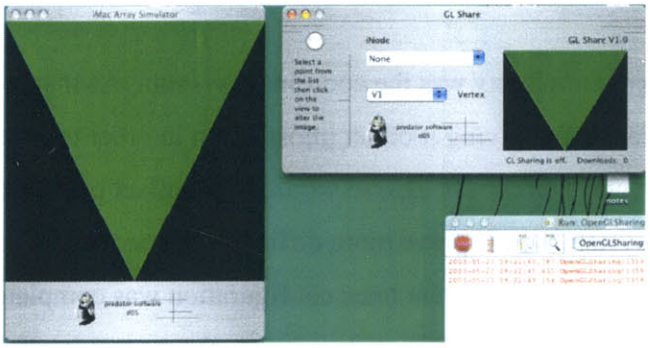


Figure 3.4: To the left is the physical array simulator. To the right is the OpenGL sharing controller.

Before embarking on building the system using real nodes, a simulator was used to improve preliminary designs and algorithmic implementation. This simulator provided 16 "screens" that represented the physical array present in the lab running in full screen mode. A simple triangle was used as the graphics object for proof of concept. Its variable parameters included the triangle's three vertices and its degree of color fade. The control center kept track of user mouse input and interaction and sent the coordinates and state over to available clients. These clients then used the data to emulate the user's desired movement on to the scaled version.

This interaction gave new life to the array. It seemed to breathe rhythmically due to the fading animation of the triangle and respond clearly to user input. It also gave the user a sense of power over something quite large and in full view of any onlookers. Furthermore, it simply worked. The only actions required by the user were pointing, clicking, and dragging.

Later, this system was augmented to handle slightly more complex shapes and parameters in order to test the mechanism of scaling. A problem barely noticeable in the simple triangle version emerged clearly. The problem involved the viewing angle on a scaled and translated graphics object. As the new complex object rotated on an axis, pieces of the object would appear on screens when they should not. Furthermore, the interface between nodes left a non-continuous gap that, when viewed up close, resulted in a non-uniform onscreen object.

The other introduced complexity was the coordinate system. As it stood, each client required a configuration file giving it offset information in order to describe its location. If the configuration grid were to change, so would these offset parameters and by hand. Thus updating the system would prove more difficult than necessary as opposed to how simple a user could control the system once configuration was complete. Although scaling was plausible within a specific grid structure, the system itself would not be able to scale as easily.

4 System Definitions

The system is described as a scalable spatially aware media sharing display system. This chapter will explain exactly what that means and present various components of the system in an effort to define terms that will be used in the rest of the document. In short, scalable increases the scope of the system, spatially aware increases the complexity of implementation, and media sharing defines the content of communication. Additionally, the system is comprised mainly of three parts: the input client, the output client, and the control center server. These three provide a basis for the whole framework of media based communication independent of platform and user location.

4.1 Scalable

Scalable is defined as the ability of the system to expand by incorporating more nodes into its network of control. In essence, these added nodes become acting agents dedicated to the system. Thus, potential output clients (see Section 4.6) that register with the control center server (see Section 4.7) become assimilated into the system as a form of output device for the control center server (see Figure 4.1). Depending on locality setup and purpose of use, the entity wishing to install the system can potentially control as many output nodes as he or she sees fit. He or she can decide to add more nodes to the network or remove them from the network. As a result, the system must be able to detect such changes and reconfigure itself without the need for administrator intervention.

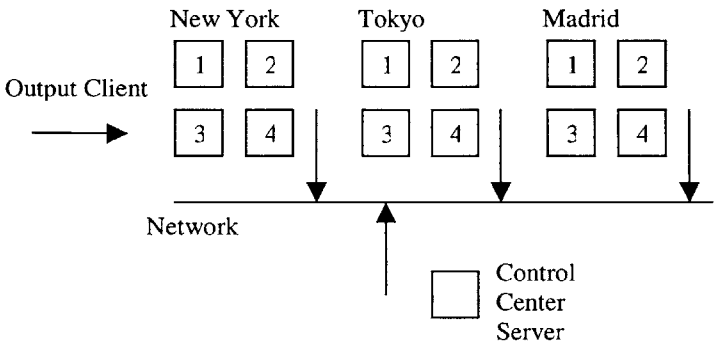


Figure 4.1: Output clients in three locations connect to the Control Center Server with the assigned identification numbers for relative spatial awareness.

4.2 Spatially Aware

Spatially aware, in this system, is defined as the ability for output clients (see Section 4.6) to know, in some sense, where they are located in relation to the other nodes on the same network. This points at an ability to cluster output nodes according to a user's input request. Thus, location specific messaging is a feature supported by the system. Location is relative compared to the other nodes (see Figure 4.1). A user setting up the system would be required to setup display nodes with a name and identification number. If the desired setup is, for instance, a 4x4 grid, each output client should be given an identification number from 1-16 left to right starting in the upper left corner. In this way, the control center server can have some sense of where output nodes are in relation to other nodes. Combined with the term scalable, the union suggests an ability to spread computation across various selected nodes and the ability to display information that is pertinent to each node. In other words, each node can potentially be accessed exclusively as single entities, single entities in a cluster, or combined to form one "single" node in a cluster.

4.3 Media Sharing

Media sharing is defined as the act of allowing multiple users access to one's media or the act of sending one's media to another. The content and methodology of communication is then scoped by this term. The system allows for communication of media types that include text, images, and video. Sharing implies the ability of users to broadcast their own media to the network and determine whether or not they would like to grant others access readily. This in turn suggests a mechanism for storage of media traversing the system. It does not, however, suggest a requirement of logging into the system as specific users. It instead suggests that any user (in this case input client) can send a request to the control center server thereby allowing ease of connection and sharing of data.

The ability to store data by media sharing involves the control center server maintaining some sort of database of transactions crossing the system and a naming scheme to protect against collisions. Furthermore, this storage mechanism should be able to handle updates. The file lookup must then be somewhat sophisticated. In order for saving and lookup to operate correctly, feedback should be sent to the user making the request. For instance, if the naming mechanism changes the name of a file sent to the system to be stored, the user must be notified so that a future access to that file will result in the correct lookup on the user's part. A possible extension to the storage mechanism is to allow for deletions as command from user space (this would require a notion of unique users, permissions, and more complex security).

4.4 Display System

First and foremost, this system is a display mechanism. The generic term of display defines a context of media independence. In other words it points to the capability of the system to display various media types ranging from plain text to custom programmable media.

4.5 Input Client

Input clients are defined as those devices making requests to the control center server touched on in Section 4.7. Input clients are also required to have a method of network communication. Although TCP/IP is the foundation, the means of connection to the control center server is device independent. Input clients can be any network accessible devices ranging from desktops to portable computing machines, even cellular phones. An interface for control, on the other hand, can be device dependent as long as it adheres to the communication protocol of the control center server (see Chapters 5 and 6). As a consequence, this allows users to implement their own control center server communication protocol compliant control interfaces. In addition, current technology in the form of E-mail can be used as a middleman between input client and control center server. Since E-mail is already heavily used in society, interfaces for E-mail are tried and

tested hence reliable. E-mail's very nature is network communication. As a result, it is an exceptional interfacing tool to the control center server and obeys a well-defined protocol that can be exploited by the control center server. It follows then that any E-mail compliant device is control center server compliant. Finally, although the system is Java-based, input clients need not run Java if adhering to the protocol. However, being cross platform, Java allows a notion of platform independence.

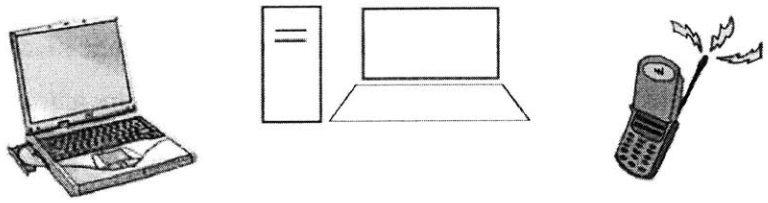


Figure 4.2: Possible input client devices

4.6 Output Client

Output clients are those devices that can be used to display various media while preserving their status as an output client. It is important to mark a distinction between output clients and output devices. Output clients have a CPU, some kind of operating system, and an ability to execute Java and QuickTime and handle network connections. Output devices can use or be used by output clients (i.e. a projector can be hooked up to a output client in order to enlarge its view).

Having the potential to run continuously for days requires a level of resource management to maintain high performance and correct output. As a result output clients must be able to reclaim resources to maintain maximum execution efficiency. This must be done as transparently as possible. In other words, a user viewing the output client should not notice a self-preserving maintenance operation executing. One approach is to have output clients shutdown and restart while masking the shutdown from an observer. Furthermore, in the event of a crash, they should be able to come back alive and regain some notion of state, continuing as if nothing had occurred. It might be helpful to think of an operating system that handles switching from process to process and masks this implementation from the running processes. Performing this task requires a sense of state

to be kept by each display client and a mechanism for restoring that state in the event of a crash or maintenance reboot.

Output clients must have some measure of configuration: at least a name and a control center with which to connect. Optional configuration mechanisms will be discussed in Chapter 5.

The sole function of the output client is to obtain commands from the control center server, parse them, execute them, and stay alive as an entity on the network (see Figure 4.1). This execution affects what the output client displays. Depending on various parameters received in the command by the control center server, an output client will know if it is to function as a single entity or part of a cluster. As a single entity, any media passed to the output client is displayed in whole on that client. As a cluster entity, only the portion that fits into its window is displayed. When viewed as a cluster entity, it is easier to visualize output clients as pixels in one display. Thus, output clients retrieve orientation information and size from the control center, performs the necessary calculations, and outputs the result as it would pertain to a pixel in the desired array space.

Output clients must also be able to handle a variety of media within the three major groups: text, images, and video. QuickTime is a partial solution to this problem as it handles many media formats.

4.7 Control Center Server

The control center server (from here on referred to as CCS) is the middleman relay station and interpreter between the two types of clients. It is the centralized source to which requests are received and parsed and from which commands are dispatched. It functions as a front end to the output clients and the message receiver for input clients. The CCS maintains some notions of state from the two clients pertinent to functionality (connection, spatial location, and identification). In order to be effective, the CCS must

run efficiently and, because of its function, handle multiple requests at a time. For efficiency, using threads to represent all connected clients and to handle the actual communication between CCS and client frees the main core of the CCS to perform more important calculation specific tasks. Without a CCS, initial communications cannot occur. The CCS must be online and must also coordinate connections as it sees fit. Furthermore, the CCS must perform various arithmetic operations and, if necessary, balance load among nodes. Finally, the CCS has the ability to store information and media into a centralized location for the purpose of sharing data as the name of the system suggests.

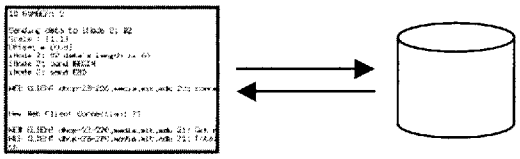


Figure 4.3: Server communicating with database to store media in centralized location.

Other responsibilities of the CCS include functioning as an operating system where it can track its state. This means it must maintain itself (just like input clients must maintain themselves). It must be able to recover from some invalid state and continue operation as good as new. One approach is to detect when the system is idle and during that time perform maintenance operations to keep the control center running smoothly and efficiently. The CCS must also cleanly close connections to clients who disconnect or crash in order to free resources and maintain efficiency.

The CCS must be able to interface with new input devices that may be developed after its creation. Thus the protocol of communication must be general enough yet specific enough to handle forward compatibility and sophisticated requests.

5 Design

This chapter will present an overview of the design of the system from a high level perspective. It will begin with a description of the iMac Array hardware and network configurations followed by a more in depth discussion on architecture. Finally, it will give a high level view of a process sent through the system.

5.1 iMac Array

The iMac array is housed in the Physical Language Workshop and consists of fourteen iMacs running Mac OS X. Each iMac in the array that runs the output client software is defined as an iNode. These iNodes are placed on two racks and are split into Sections S1 and S2 due to the physical limitations of the hardware configuration. The iNodes themselves are wired together via Ethernet cables and switches. Switch 1 connects all of side S1 and Switch 2 connects all of side S2. One iNode from side S1 is attached to Switch 2 due to port limitations on the switches. Both switches are linked to the World Wide Web and the local area network thus creating the networking context. Two switchers are used to independently connect sides S1 and S2 with appropriate I/O devices. Finally, a wireless mouse and keyboard exists for each side and is synchronized with the switchers.

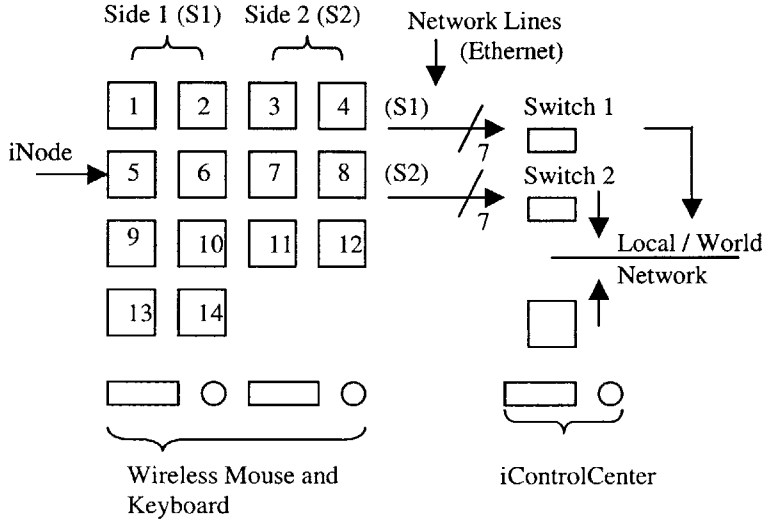


Figure 5.1: The physical setup of the system. The array to the far left is composed of the iMacs turned iNodes. All of the iNodes connect to Switches 1 and 2. These switches in turn connect to the network allowing for communication with the iControlCenter.

5.2 Software Architecture

The system is designed as a 3-tiered client/sever model separating application users from output nodes. Figure 5.2 depicts these tiers and their connections without the smaller details. The nature of the system points to this model as an obvious choice and includes remote multimedia processing and computation. The input clients and output clients communicate to the CCS via TCP/IP and socket connections. On a LAN, this communication occurs with increased speed (depending on the LAN Ethernet connection).

The means of connection for both input clients and output clients, as previously stated, is device independent. However, for input clients, the interface of control can vary. Since a common communication protocol has been defined, any input client that adheres to the protocol can utilize control interface implementations spanning many designs. Within this architecture three input clients have been defined: the web client, the CCS Control Client, and the parallel client. The CCS Control Client is a special purpose client allowing GUI access to specific CCS operations. By fitting cleanly into the communications protocol, E-mail, another clear interfacing tool, functions as yet a fourth input client. Since it does not directly connect to the system, E-mail, like the CCS Control Client, is a special purpose input client. The system instead connects to E-mail repositories.

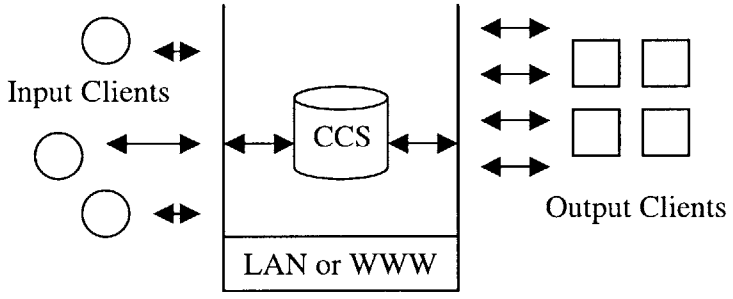


Figure 5.2: A high level view of the 3-tier model. It consists of both input clients and output clients communicating with the control center server.

The output clients, although adhering to a similar protocol, exercise less freedom of expression. Since these clients receive instructions from the CCS, they must be similar to a degree. This design decision sacrifices varied compute client interfaces for ease of communication and instruction execution. However, if new types of output clients are

envisioned, they can be added to the system as long as they obey the foundational interface of an output client. Thus, extensibility is possible. In this architecture, the output client is the iNode client and primarily serves the function of display.

The CCS is a multi-threaded server that handles both types of client. The CCS must be efficient only in order to be effective in handling multiple requests at any given time. Thus using threads allows computationally intensive tasks to be pulled away from the core functionality of the CCS. Figure 5.3 depicts these threads as they pertain to the connections (yet more threads exist and will be discussed in Section 5.3). Input client threads and output client threads handle the actual communication between the core control center server and the respective client types (although all clients initially connect to the core CCS, they are subsequently redirected). Furthermore, there exists a "store" which is used to save media data. The client threads handle maintaining certain state. In addition, some have persistent connections while others have transient connections. On specific occasions, these threads themselves can connect to each other in order to communicate to actual clients outside of the control center server black box.

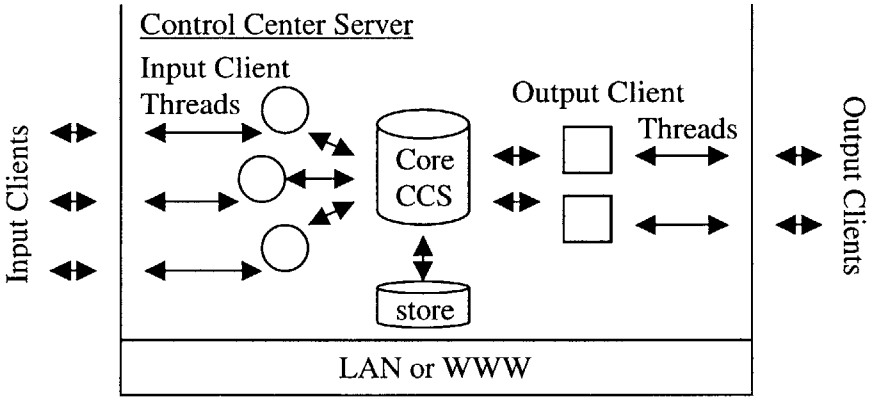


Figure 5.3: This is a detailed view of the control center server. Notice that the CCS is actually composed of distinct parts. The input client threads handle some front-end communication to actual input clients while the output client threads do the same to the actual output clients. The core CCS communicates with these threads once they have validated incoming data and uses the information to perform a task. The media data store communicates with the main core.

The client threads will receive the data from a communication pipe and perform validation. If the data is valid, these clients can add the command to a queue where the

main core of the CCS can retrieve and process them. This way, tasks can be handled in parallel and relieving other parts of the CCS. A form of fault isolation is achieved by this mechanism reducing the chances for single points of failure. In addition to relieving load, separating the CCS into these parts allows for a form of fault isolation to surface reducing chances for single points of failure. If an input client goes down or its respective thread bottoms out, it will simply cease to exist as far as the core CCS is concerned. The same is true for the output client threads.

Furthermore, this design allows the infrastructure to grow in output power via the addition of potential output clients. Where parallel computation is concerned, although the overhead of communication and output client data managing increases, the speedup possible given the extra processors should mask such drawbacks. Two special purpose threads that handle the CCS Control Client and the Mail Client respectively accompany the two distinct client threads pictured in Figure 5.3. The CCS returns messages to all input clients and special input clients via thread equivalents. Input clients can optionally choose to receive them. The output client threads are used to dispatch parsed requests turned command to the actual output clients.

In supporting many output client connections, the CCS utilizes a data structure that can differentiate between specific clusters and random nodes. Because of this, output clients must register with the system in order to function as output clients. Specifically they need at least a name as identification. Those with only a name will be committed to the general store of available output clients with automatically generated identification numbers. Those specifying a group name will be identified as such and the ability to scale media across multiple screens of that group is granted. In addition, these grouped output clients need to be assigned rank by a user. The CCS will use this rank information to infer spatial structure. A final configuration requirement is the grid structure. Out of all the nodes in the group, at least one must have a short description of the grid structure of the group (i.e. 4x4).

Since the CCS also handles storage of media, issues involved in this procedure deal with name collision and file lookup. Name collision will occur when input clients request some media to be stored whose name already exists in the shared store. As a result, the system must decide whether a file new altogether or is to be overwritten or updated. If the file is misconstrued as something it is not, a request for the file to be displayed later could result in the wrong lookup. In this design, the CCS never overwrites a file but appends a number to the name, saves it, and sends this new file name back to the requester. This way the requester knows how the file was saved and can thus request the correct file to be displayed at a later time with the correct name.

Two communication protocols descend from a common protocol while a third currently does not. The types of existing protocols are the special client protocol, the web client protocol, and the parallel client protocol. The web client protocol is more aptly described as the "regular" client protocol. Regular clients are those input clients having java capabilities while special clients are those that do not. Special clients include microcomputers, microcontrollers, and other low level electronic devices that may desire to communicate with the framework and/or control the system. Once received, this command is parsed and wrapped into the form of the web client protocol. Both the web client protocol and the special client protocol share common attributes.

The web client protocol and parallel client protocol define two new sets. The web client protocol defines all clients using the system for display of media. The parallel client protocol defines all clients attempting to use the system for parallel computation. These two protocols differentiate via a type field. The parallel client protocol, in its current implementation, is specific to a certain type of application (as well as the other parallel aspects of the system), though it can be generalized for more tasks (discussed in Chapters 7 and 8). The details of these protocols will be discussed in Chapter 6.

Designing the aforementioned protocols in this way allows for multiple device types to connect to the system and use its resources for gain. Furthermore, the web client protocol allows for complete *objects* to be passed between input client and CCS, and CCS and

output client. This is deliberate in an effort to allow custom media types as well as general parallel applications to be distributed via input client to the CCS and finally to targeted output clients. This concept will be further discussed in chapters 7 and 8.

The appendices contain detailed diagrams of the software architecture including connections, packages, and various subsystems in relation to the entire structure of the system.

5.3 High Level Description

In typical fashion, a user of the system would start up a client application and submit a request. Supposing the device were a smart phone contacting the system via E-mail along with an image attachment, the system would respond by first opening the E-mail and re-representing its contents as a Mail Message (see Chapter 6). Once all pertinent E-mails are similarly dissected, the mail client thread will proceed to parse the body of the message and save any valid attachments to a store. Valid attachments are those that are QuickTime compliant.

The body of the E-mail contains the instructions that will direct the CCS. This command must comply with a request protocol. The request protocol is lenient and supports many options. Supposing this request is simply to save the attachment into the store and post the media across a full grid, the command would contain, at minimum, a group name and the word "all".

Once the command is successfully parsed and validated, it is submitted to the web client protocol generated along with an internally re-generated URL to the desired media and, if applicable, a username. The output is then added to the CCS command queue and notified for processing by the main core.

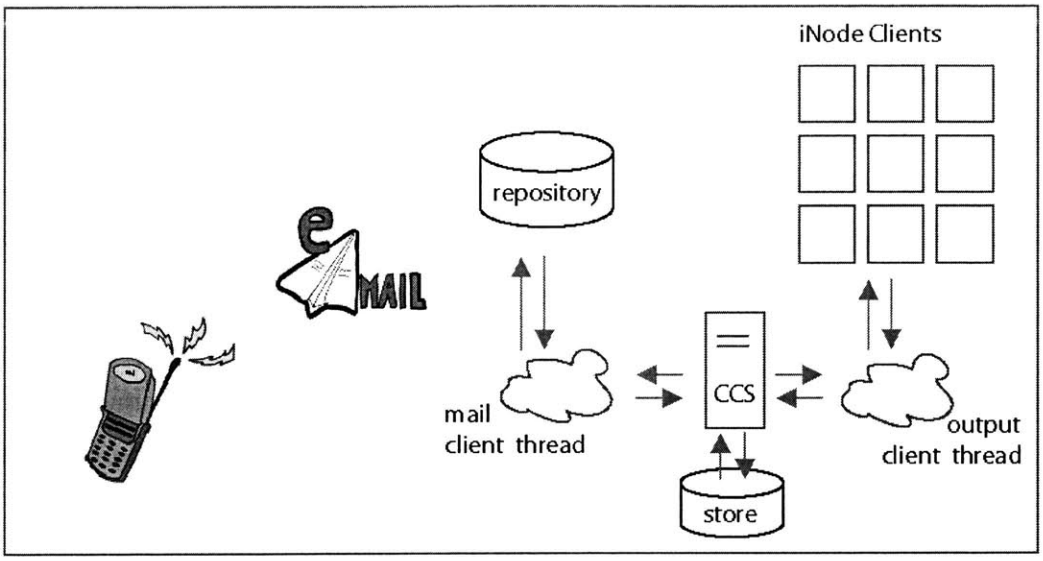


Figure 5.4: Visualization of the system at work from a request via email.

The CCS, utilizing a notify-wait mechanism, is either processing current commands or is awakened by the admittance of a new command. The CCS begins to pick apart the request while setting up various flags in preparation for display by the targeted nodes. First the CCS determines whether or not the command is a simple load request to save the contents of the media at the specified URL to the store, or a display request. Once established, the CCS then ascertains whether or not the request is targeted towards a specific iNode Cluster group connected to the array or to a general output client. Discovering that the request is targeted for a specific group, the CCS checks its iNode Cluster Map table for the desired group. The nodes of this group are represented as a red black tree. Since the command issued targeted all nodes, the CCS employs the CCSProtocol for inferring the structure of the cluster along with the relative scaling and offsets required to fit the display specification of the command. The CCSProtocol packages this information and returns the completed minimal calculations to the CCS. Since the CCS recognizes the command as targeting the entire cluster, it initiates an optimized routine for dispatching the node specific instructions to the target cluster.

After dispatching the instructions, the CCS will continue parsing the queue if more instructions exist. Otherwise the CCS will wait patiently for a new command. If the idle state is reached, the CCS will initiate the history process that executes ordered previous display commands.

Upon data retrieval the output client, also utilizing the notify-wait scheme, will determine whether the incoming instruction is for display or for parallel computation. Attributing the command to display, the client will extract the minimal position instructions (an integer offset and a integer scale) as well as the data URL. This information is then submitted to the parent GUI for processing.

When the parent GUI acquires this data, it utilizes the procedures of its internal view object to retrieve the media, properly scale and translate it to the specification, and update the view on the actual display.

6 Implementation

This chapter will discuss the major components of the system in more detail. Without these components, the tasks and goals of the system could not be met. Included in this discussion are the communications framework, control, message handling, and pertinent data structures and algorithms. Some parts of the parallel extension to the system are discussed as well.

6.1 Communications Framework

The communications framework is based on a class called GeneralComm. The necessity of GeneralComm came out of a vision to send custom packaged media, such as Java objects, through the pipeline and the emergence of the parallel image compressor extension. As not to limit the ability of devices to connect to and control the system, the previous pipeline involved byte streams. This original pipeline still exists in the form of PatInputStream and is authenticated by the system using the special client protocol. The pipeline exists for multi-device compliance and for converting commands into byte format to save on disk as history.

The GeneralComm itself is implemented as an Object Input and Output Stream designed to partially abstract away the problem of writing and reading between connected components. By relying on predefined Java components, its chance of failure is minimal.

To further abstract away the problem of making connections, maintaining those connections, and automatically retrieving information from these connections, BGListener and BGDownloader were developed. Both utilize the GeneralComm communications channel as a foundation. BGListener takes as input a hostname and a port. Optionally, it can take a buffer in which to place received data in first come first served order. This class can be set to connect and re-connect indefinitely or a specific number of times. With the creation of this class, the problem of communication between

clients and CCS is dramatically reduced. Appendix A details these framework connections.

Protocol Parameters

Parameter	Value
group	any string
load	true/false*
cluster	true/false**
names	target names
ids	target ranks
data	text or URL***
user	a username

Table 6.1: This table lists the possible parameters and values that can be used in commands to the system as well as input parameters to the protocols. In addition, values can be omitted for default behavior.

* for email: load/display
** for email: cluster/single
*** or objects

The rest of the framework revolves around the protocols developed for communication. For the display system, a user has the options of loading or displaying, scaling, selecting multiple targets, and selecting a group (see Table 6.1). All requests must, however, contain data that could be defined as a URL to specific media on the web, the name of a file already in the systems local or remote store, or some text for display. With this in mind, the protocols were designed to handle all possible valid permutations of this type of request. Thus web client protocol and special client protocol are similar in this respect, along with the imposed limitations I have put on the system as far as node count possible per group is concerned. Web client protocol is more flexible in transmission than special client protocol since it utilizes abstracting java elements. The special client protocol, on the other hand, had to be designed as a header and footer type packet with data located in the center (see Section 6.4.4)

The CCSProtocol was developed as the solution to supporting, in theory, infinite iNode connections to the system for scalability. The hard coded knowledge of the grid housed in the Physical Language Workshop was stripped from both the input clients and the web and special client protocols in favor of a redesign that would dynamically infer the grid structure of connected iNode clusters. This design also moves the burden of scale and position calculation from the clients back to the server.

The parallel client protocol is currently specific in nature and serves as minimum validation for data packages (see Section 6.4.8) that are sent through the system.

6.2 Control

There are two layers of control associated with the system. The first is user related while the second is administrator related. A third possible layer of control is associated with iNode Cluster management.

User related control involves users attempting to control the system via input clients. The only measure of control afforded is the ability to make output requests that the CCS will validate, augment, and dispatch. There exist seven control fields that users can manipulate in order to create a desired outcome (see Table 6.1). Furthermore, users have the freedom of utilizing any media type supported by QuickTime.

Administrator related control focuses on minimal maintenance of the Control Center Server itself. It can be controlled via command line scripting or a GUI tool. Control is limited to starting up the server, shutting it down, setting debug modes, and setting idle time and history dispatch execution periods. The GUI and the command line tool allow remote access to the Server, however, initial launching of the server must occur locally on the machine the CCS is installed (or via remote login).

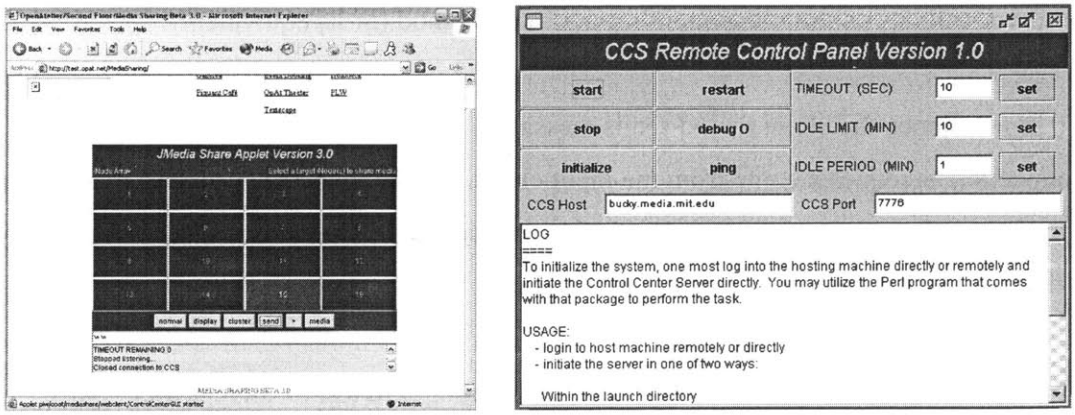


Figure 6.1: A simple input client control interface and the Control Center Server remote access control interface

Base versions of these control interfaces allow for motivated individuals to create more advanced control mechanisms for functionality not implemented.

With regard to iNode Cluster management, this user need only install the software and ensure the correct minimum configuration for the desired type of cluster (see Chapter 5 Section 2) as well as recovering any nodes that have failed beyond its self-preserving measures. Configuration takes the form of a text file with one line of parameters separated by the colon character. A sample configuration follows.

SOMEHOST.MIT.EDU:PLWGRP:PREDATOR:1

The above configuration would tell the output client software to connect to the host SOMEHOST.MIT.EDU and register this node with the group PLWGRP, the node name PREDATOR, and the node rank of 1.

6.3 Message Handling

All components of the system that receive information have some form of listening mechanism accepting packages. Most of the major components utilize the BGListener to encapsulate this operation. As packages are downloaded, they are added to the end of a message buffer. This buffer has an observer checking to see whether data is present. When this buffer is notified of new data, the observer wakes up and, if data is present, the observer calls its parent component's parseData method (or its equivalent).

To input clients, the client thread sends messages to reset the timeout counter on the connection as requests come from the input client. This allows the system to maintain a semblance of a persistent connection during transactions. These messages are caught by a listener and handed off to correct procedures.

To output clients, the client thread sends ping messages to test its alive status. Since output clients are persistently connected, the CCS needs to assure the integrity of that

connection. Conversely, output clients need to ensure the integrity of the CCS though in less periodic intervals by attempting to send alive messages. If these fail, the output client closes its connections and attempts to reconnect at given time intervals. Messages sent to the output client are caught and handed off to a primary stage handler that identifies the type of message for further processing by appropriate operators.

In the CCS, there are two levels of handlers. Still, respective client threads first receive all messages. The threads use the correct protocol rules to perform a minimum data validation operation on the message. Once cleared, a new message is created with augmented information and added onto the command queue of the CCS. The second level involves the CCS core operating on the new command. A simple buffer observer handles this dispatch calling upon CCS methods that identify command type and enact proper measures (see Appendix A).

The parallel extension to the system provides dedicated handling to its clients. In the input clients, the message parser extracts information from packages according to a phase number. Specific phase numbers flag different tasks. For the most part, the tasks involve sending regular files, and receiving compressed files. Other tasks involve maintaining client connections and status of the current job (see Section 6.4.8).

In the parallel extension to output clients, a two-stage message handler is required. The primary stage involves identifying the type of package received. If the package is a parallel task, the second stage message handler is created. This second stage handler is required since output clients, upon receipt of a valid package, make a direct connection to the input client for file transmission. Parallel data handlers receive their first package from its CCS output client thread counterpart. The package contains the client's allocated part of the load to compute, the host name and port with which to connect, and load assignment (see Section 6.4.6 and 6.4.9). The handler then makes the connections to the input client host. Most of the subsequent packages received will be from the input client. The only case where a package will not be from the input client is if the input client submits an exceptionally large job to the CCS or if many input clients are making

requests thus flooding the system. In these cases, the CCS must peter out the load as output nodes become available.

The parallel extension to the CCS adds a more complicated measure of message handling to parallel tasks. Once the second level message handling mechanism (discussed earlier) identifies the command type as parallel, the task is added to the new job thread queue that observes its own state. This new job thread queue parses the data and retrieves the compressed load tree. This tree contains size information and identification numbers to denote directory structure (see Section 6.4.6). The size information is used to calculate an overall equalized load on the output clients. If a job is too large or the nodes are overloaded, this job will have to wait and its contents petered out as nodes become available. The packages are actually handed off to the output client threads that will handle further communication with the output clients (including status updates)

6.4 Data Structures and Algorithms

While developing the system, various special purpose data structures were created in order to represent data and/or facilitate efficient procedures later on. These structures include the Media View and Media Presenter, Red Black Node, iNode Cluster, the Red Black Load Tree, Load Tree and its subsets, the parallel data packages, various lookup tables, and length encoding among other things.

6.4.1 Media View and Media Presenter

The Media View and Media Presenter classes are used as the display engine for the output client display. The media presenter handles most, if not all, of the QuickTime communications. It is specifically made to render text onto a graphics context. Currently it supports regular text, the newline character, and smart sizing. Smart sizing is the ability of the media presenter to center and approximately size the text to engulf as much of the display space as possible. The media presenter in theory handles all media types supported by QuickTime. Using QuickTime as the backend render engine is deliberate as it provides a semblance of media independence.

The media view uses the media presenter as its graphics context for presenting various media types. Once the graphics context is created, the media view performs the task of scaling and translating the data according to the current screen size and the issued display size. The screen size is representative of the current window size of the application. The display size is actually a multiple of the screen size in both the x and y directions representing the desired full display size of this view. The media view maintains the aspect ratio of the media and scales it to the maximum size that allows all sides of the media to fit into the display space. Clipping is handled by the QuickTime engine and occurs only when the desired display size is greater than the screen size. Any part of the image outside of the screen window is clipped.

6.4.2 Red Black Node

The red black node is the component that makes up the red black tree data structure. The red black tree implementation used in the control center server is derived from Cormen, Rivest, Leiserson, and Stein's Introduction to Algorithms design [37]. The algorithm ensures a balanced binary tree structure amidst addition or subtraction of nodes. This also ensures that queries into the tree will take time in O(log N) where N is the size of the tree. The red black node in this system has been augmented to store as many elements as it so desires. A mechanism for copying its contents to another node was also inserted into the red black node implementation.

6.4.3 iNode Cluster

The iNode Cluster is an abstraction used to facilitate organization of iNodes into groups as well as speed up access to these iNodes. The iNode Cluster is an extension of the red black tree implementation. It is identified by both a name and an identification number. The name refers to the user assigned group identification of a physical iNode cluster onsite. The identification number is a unique identifier assigned by the control center server. The grid structure of this cluster is also stored within the data structure.

The iNode Cluster houses all connected iNodes connected. An iNode map is used to map an iNode's name to its rank. The rank itself is used to create a new node in the tree structure. Ideally, a red black node in the tree would be an iNode itself, however the red black tree implementation makes this possibility difficult. One of the methods for removing a node in the red black tree algorithm by Cormen et al involves copying the state of a node that will assume the empty position to be into the node that is to be deleted. In this way, removing the node from tree becomes more efficient. Since iNodes have complex state, the copying of that state proves difficult. As a result, the red black nodes in the iNode Cluster implementation are those described in Section 6.4.2. An iNode identifies the red black node with its rank and becomes an element of the red black node (see Figure 6.2).

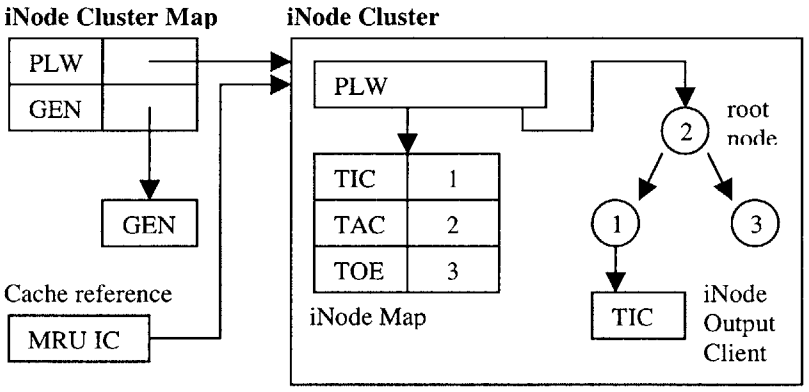


Figure 6.2: CCS iNode organizational structure. The cluster map points to actual clusters that in turn reference iNodes in a balanced tree structure.

The iNode Cluster provides methods for manipulating the tree itself. It supports additions and subtractions of iNodes as well as searching. Status methods are available as well including the ability to close all iNode connections. When adding an iNode to the cluster, the data structure cleaves grid information from the iNode if present. The iNode Cluster then maps the iNode name to the iNodes rank for later lookup and inserts a new red black node with the iNode as its element into the tree using the iNode rank as the key. If, for some reason, the iNode already exists in the tree, a name collision is issued and the current iNode is removed and the new iNode inserted. An iNode can be removed either by using its name or its rank. If a name is used, the rank is determined using the iNode

map. Once rank is determined, the data structure searches for the node in O(log N) time, closes any open connections to remote iNode clients, and removes the node from itself.

The iNode Cluster itself is referenced by an iNode Cluster map within the CCS. All groups connected to the system are stored within this map using the group names as a key to the iNode Cluster data structure (see Figure 6.2). Since multiple requests can be made on a group in succession, a cache mechanism is in place keeping track of the most recently used iNode Cluster group. If the group referenced by an incoming command matches the group name of the most recently used cluster group, no cluster group look up is required eliminating the unnecessary overhead of retrieving name to cluster mappings for every incoming command.

Finally, any incoming iNode connection that has no group is assigned to the general group cluster located inside of the CCS. For purposes described in Chapter 7, the identification numbers assigned to these iNodes by the CCS begin after the group rank limit of defined group nodes. A background process that removes all inactive nodes from all iNode Cluster groups checks the status of all iNodes periodically.

6.4.4 Protocols and Grid Translation

The design of the protocols circle around the valid inputs into the system touched on in Section 6.1. Table 6.1 details these valid inputs. The particulars of the extended protocols' designs are described below beginning with the special client protocol.

The special client protocol is structured such that all pertinent data is enclosed between the fixed length start and end codes. This way the start and end of a packet can be determined while reading a socket. The length measurement signals the PatInputStream with the number of bytes after the start code to expect including the end code. This speeds up the process of retrieving the data from the network. The length is measured with three bytes. The second byte in the length measurement is fixed at 0xFF. The first byte is a counter indicating how many multiples of 0xFF exist in the size of this packet.

Finally, the third byte acts as the fine grain size measurement for preciseness (see Figure 6.3). The rest of the protocol includes various counters and length measurements regarding other data stored within so that the special client protocol can correctly extract the required information from the packet.

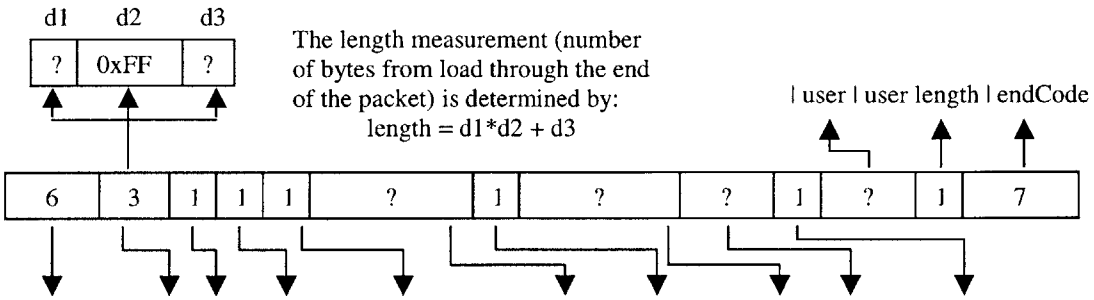


Figure 6.3: The specific protocol associated with the special client protocol

The web client protocol is a much simpler protocol that uses a map to coordinate data with specific keys and the GeneralComm communication channel to handle "packets". The web client protocol has a key for iNode names. This is in reference to those iNodes that do not belong to a user defined iNode group. As a result, these nodes do not have user-defined ranks and can only be targeted using their names from the user's perspective. The web client protocol can also be used to convert data to or from the special client protocol.

The CCSProtocol is a special purpose protocol used by the core of the CCS to infer grid structure and relative scale and offsets for media data. The scale is more accurately described as span measure that is a function of grid blocks. All iNodes in a cluster are viewed as grid blocks. When a request is sent to scale media across x number of iNodes, this number is broken down into units of grid blocks down and across. The CCSProtocol harbors two specific data structures for translating a scale request to span coordinates and specific iNode grid offset coordinates: the iNodeCoordConversion and the iNodeMatrix.

The iNodeCoordConversion is lookup array that uses an iNodes rank to determine its location in grid coordinates. Its size is equal to the total number of iNodes in a grid

multiplied by two (since each node maps to two coordinate numbers). The array is structured so that the row and column coordinates are indexed by the rank number and the rank number offset by the total number of iNodes in a grid respectively. Since rank number begins at one and grows positively, the index measure must be decremented by one to point to the correct spot in the array (see Figure 6.4).

The iNodeMatrix is a three dimensional array used to model a group's grid structure. Three elements exist at indices (row,column). These are the *x* and *y* relative offsets and target flag. If an iNode at index (row,column) is selected, its flag will be raised and relative offsets computed. All offsets are relative to the most minimum target iNode selected. For instance, if iNode 6 in a 4x4 grid were the most minimum selected, its offset would be (0,0) while all others would be relative to iNode 6 at (0,0).

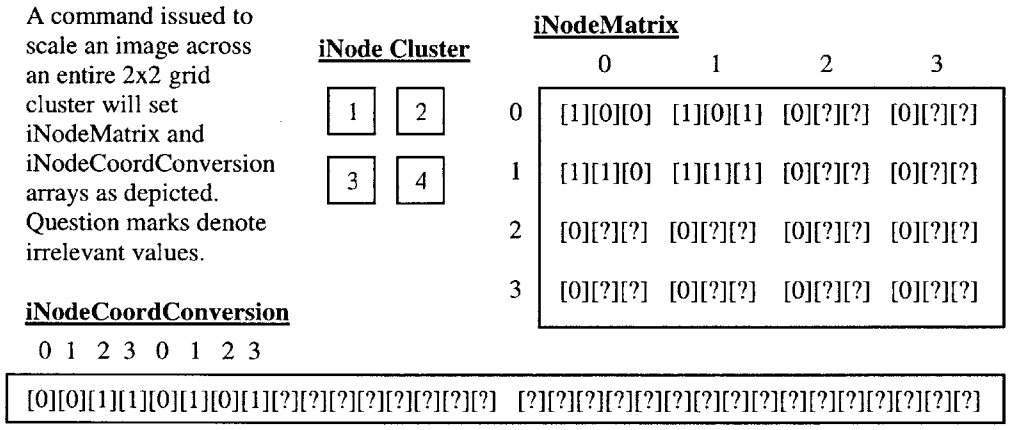


Figure 6.4: Example settings of the iNodeMatrix and iNodeCoordConversion lookup arrays

Both the iNodeCoordConversion and iNodeMatrix lookup arrays are initialized to a size 4x4 grid. This way both arrays can be reused for multiple cluster groups without having to be reallocated. The iNodeCoordConversion has an extra requirement that if the actual row and column configuration of an incoming cluster group differs from the most recently used grid structure the actual values for each index of the area must be re-calculated. Even if the size is the same, these values must be recalculated since the coordinate conversion of a specific iNode will be different for two differing configurations. Now if the overall size of a grid cluster is greater than that already

located, both lookup arrays must be reallocated to support the new greater size. This design benefits by reducing the need for recalculation making reallocation a rare event.

The parallel client protocol is present mainly to require a phase number, a load tree, a host ID, and a port. The protocol is also present to validate packages being sent across the communications framework. Section 6.4.8 covers these elements in more detail.

6.4.5 Mail Message

The Mail Message is a data structure used in the Mail class I created for logging into an E-mail store and extracting data from E-mails fitting a certain search criteria. The Mail Message itself houses the extracted data and wraps the information. Figure 6.5 depicts the structure of a Mail Message. The main contents of a Mail Message include the message ID, the E-mail address of the sender, the personal name associated with that address, any attachments associated with the E-mail, and the body of the E-mail. A Mail Message can accommodate more attributes if necessary. By constructing these Mail Messages, all pertinent and valid info can be put into one buffer where each element represents a new set of user commands. The buffer can then be batch processed.

Mail Message

MID	message ID number
ADDRESS	address
NAME	personal name
ATTACHMENT	attachments buffer
BODY	body

Figure 6.5: Mail Message object and its relation of parameters to content. It also supports user defined attributes.

6.4.6 Load Tree

The Load Tree data structure is used to encapsulate directory structure and file information on the input client machine. Two of these have been developed: a Directory Load Tree and a Compressed Load Tree. The super class of these two (the Load Tree) provides functionality common to both encapsulating data structures. The difference between the two is the form the data takes within each tree.

The Directory Load Tree is defined as a tree whose nodes represent child directories and are identified both by an ID number and a directory name (including its path). Files are encapsulated as a string concatenated together with their specific file size. Other information in each node include the number of files within that directory, the total size of these files, the number of immediate child directories in this directory, the total size of the files in these directories, the total number of subdirectories below this tree (how many directories recursively living within this directory), and the total size of all subdirectories including the total file size of the top level directory. Since these sizes can be exceptionally large, the base 2 logarithms of the sizes are encoded instead. Furthermore, a quick lookup table is compiled that uses the ID numbers of the directories as keys and the paths of the directories as values. None of the nodes in this tree contain this lookup table. It is only dynamically generated with the tree and handed off to the caller. This lookup table is later used when requests from the compute client begin to arrive. These requests only use integers to identify directory and file structure. Finally, a depth integer is used to control how deep into a directory structure this tree should replicate.

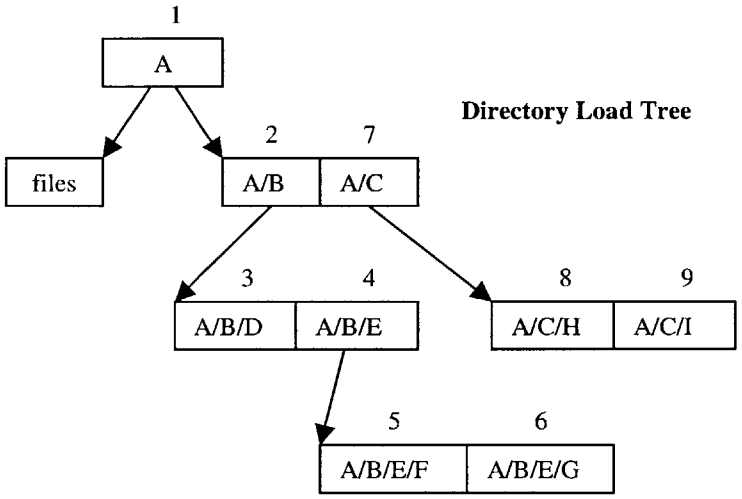


Figure 6.6: Structure of a directory load tree. This one has a depth of 3. All directory nodes point to a files data structure (although only one is pictured). The bottom level is not expanded (depth 3 limit). Conceptually, a compressed load tree is similar only with the integers as shown.

The Compressed Load Tree is defined similarly to the Directory Load Tree. However, instead of containing any string information, all data are encapsulated in integers. All nodes have the same ID numbers as the Directory Load Tree. Files are encapsulated into a simple integer array whose indexes represent the size of specific files and value denotes the number of files of this size present (effectively encoding information in smaller

space). Also, this method of compression allows for fast lookup. Finally, a lookup table is created using the directory IDs as keys and the encoded file arrays as values. There also exists a special NODES key whose value determines the number of directories present (see Figure 6.6). The table itself can actually replace the compressed load tree and sufficiently encapsulate required data. Figure 6.7 is similar in representation to this table.

Both of these trees are sorted according to size order from smallest to largest and left to right (although directory structure is maintained). This is important for faster processing of data on the CCS side and creating convenient invariants. For instance, an output client can quickly calculate various size information by looking at the file array, accumulating the count of files whose sizes equal some threshold, and then requesting files "i" through "j" from the input client. Since everything is in sorted order, the input client can simply use its own array and collect the files "i" through "j" into a package and send them. The directory path lookup table's function now becomes apparent. When an output client sends a request, it sends directory IDs and corresponding integer file ranges. The input client then uses this ID number to quickly select the directory where the files are stored and package them up.

6.4.7 Red Black Load Tree

The Red Black Load Tree is a data structure used in the parallel extension of the CCS to keep track of current load on available output clients. This structure provides methods for balancing the load across all output clients and assigning specific load and tasks to specific output clients. The red black load tree utilizes the Cormen et al red black tree implementation. The keys of the tree nodes refer to a load threshold while the values represent all output clients currently at a certain load threshold. Instead of holding the actual output clients, these values are the unique IDs of the output clients. The method of load balancing and concepts of load threshold are further discussed in Section 6.4.9.

6.4.8 Data Packages

Data packages encapsulate various identification information and instructions to be carried out. They also contain various messages important only to the receiving component. For the parallel context, these packages differentiate according to phase. There are seven phases (seven, being the number of completion, denotes the completion of a full job request). Each data package contains a TYPE and PHASE field for identification purposes.

Phase I packages are sent from the input client to the CCS and identify it as a parallel client requesting a job. It contains the compressed load tree, its host ID, connection port for receiving of thumbnails, and the save location.

Phase II packages are in two parts. One is from the input client thread to the output client thread via the core CCS message handling mechanism, and the other is from the output client thread to the actual output client. Phase II A contains, in addition to the original Phase I package, the actual input client for later communications from the CCS. Once the new job queue thread handles the message, the package no longer contains the reference to the input client thread, but does contain the load information for specific clients including coupled data structures containing both an array of directory IDs and file number and size integer arrays related to allocated tasks (see Figure 6.7).

Directory/File Assignment

dir	[2][5]
2	10 11 12 13 16 17 ...[5][6][2][1]...[2][1]
5	8 9 10 11 ...[18][8][2][5]

Figure 6.7: This data structure encapsulates assigned directories and assigned files. The numbers above [x] is the index of that value and represents a threshold value. The value "x" itself represents the number of files at that threshold.

Phase III packages are sent from the output clients directly to the input client after making a connection request. This package contains, besides the default information, the directory IDs and file integer arrays the output client wishes to first download and

compress. The output client sends this information along to the input client who responds in kind.

Phase IV packages are sent from the input client to the output client and contains Host ID, port, and file paths and names along with their corresponding images. These data should match that requested by the output client.

Phase V packages are sent from the output client to the input client and contains the compressed image files with the same file path and names used as keys. The input client then saves these files to an earlier user specified save directory.

Phase VI is an output client status update that is sent to its output client thread representation on the CCS. Phase VII, as mentioned above, signals job completion.

It is important to state that most (if not all) of the data package message passing occurs in the background. Thus, all of the initial packages appear to have top priority since no other computationally intensive tasks are being undertaken. However, once actual image information is sent, computationally intensive tasks emerge and the processor on each machine must be shared with sending and computing. Thus, by sending them in the background, the computationally intensive tasks can run "seamlessly".

6.4.9 Load Balancing and Length Encoding

Load balancing occurs on the CCS by through the red black load tree referred to by the new job queue handler. The balancer uses the following simplified equation to balance the load among output clients:

$$X_1 + X_2 + X_3 + \ldots + X_n = T$$

In this equation, *T* denotes the total load and includes the sum of the current load over all output clients and the new load about to be introduced. The subscripting *n* terms denote

the threshold level where n is the exponent of a base 2 number (i.e. 2^n). The X_i terms denote the number of nodes at threshold "i".

When the base 2 logs of the original number are expanded the equation works by summing all load together into a new total load using the thresholds as the factor in deciding the current output client loads. Next, we divide this new load by the total number of available output clients. This new number is the load that each client should now possess (including their current loads). So, for each node, we subtract the threshold they are currently categorized in from the new load, then allow the red-black load tree to reassign the output client to a new threshold node. The red black load tree uses the expanded sizes to compute these load sizes in order to precisely keep track of how much load is *leftover* after assigning as much load as possible. To get the maximum amount of load assigned on the first try, the algorithm begins by first assigning the largest file possible to the most loaded node possible. This results in maximizing the most loaded node, then the next most loaded node, and so on and so forth until most, if not all, of the job is assigned. By assigning load in this manner, all of the biggest files are sure to be assigned immediately using the smaller files as filler on a per node basis (like putting the biggest rocks into a bottle first and then filling all the cavities with sand).

It is important to note that by encoding size information using logarithms, actual size is approximated as a maximum bound. Since these are multiples of 2, the approximation can become exceptionally large as file sizes grow in size. For instance, a file of size 12 and a file of size 31 must be given a total size of 32 -- significantly larger than the actual size. In practice rarely do image files reach greater than 200 megabytes in size for conventional use.

6.4.10 Thumbnail Generation

The actual process of generating thumbnails turns out to be the simplest part. First, it is embarrassingly parallel. Since this is the case, potentially any fast thumbnail generator could be used and simply called by the Java program on the data. I first used JAI (Java

Advanced Imaging) that utilizes native code for faster image processing; however this package will not run on the iMac computers until an update to Jaguar is made. However, the ImageIO library of the Java 1.4 distribution can also perform image processing, albeit slower. The chosen method of generating thumbnails is irrelevant since the actual task of thumbnail generation functions solely as proof of concept.

7 Usage

During the development of this system, various usage scenarios were envisioned and some embarked upon. Furthermore, there exist current extensions to the system (in part or in whole) that have been used for some of these scenarios or otherwise. This chapter will present these findings.

7.1 Scenarios

In this section I will discuss various usage scenarios for this system as it pertains to business, recreation, education, and as an art medium. It is important to note that output clients, along with the possibility of being simple desktop computers, can be laptops and tablet PCs. Any device that can run Java can potentially be an output client. It is possible, then, to attach any display enhancing device (such as a projector) to output clients that support such connections. For example, a small tablet PC can be used as a dedicated output whose visual area is enhanced by a projector. The scenarios discussed below are a small subset of many possibilities and areas of society the system can be used.

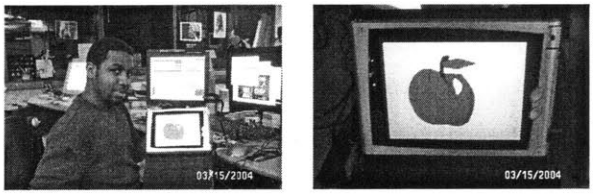


Figure 7.1: Mobile tablet PC hookup as output client device.

7.1.1 Business

A key aspect of business is fast, efficient, reliable, and informative communication. In a business, communication takes place in various forms such as meetings, conferences, design reviews, and memos. From experience, it is observed that many companies use E-mail as the messaging tool of choice. It is quick, effortless, and fast. E-mail has the added benefit of having text and attachments in one bundle and giving the recipient the option of immediate or delayed viewing.

The Media Sharing system does not seek to eliminate E-mail as the messaging tool of choice. Instead, it seeks to compliment it as a messaging tool for urgent and immediate information. It has the added benefit of displaying any media such as design drawings or concept drawings that might need immediate viewing. Furthermore, if text messages must be sent and seen the very moment they are issued, this system displays them immediately. By smart sizing these messages, short ones will appear large and long ones small.

Besides serving immediate messaging tasks, the system functions well as a tool for presentation amidst design reviews and conferences. The ability for the system to support many users without need for setup or configuration is desirable when time is a commodity that cannot be wasted. Design reviews and meetings go long enough and trimming down the time required in these meetings is a cherished idea. Electronic presentations involving images and video can be queued up with ease and presenter switching time reduced. As a result, more time will be spent delivering technical information rather than setting up.

Companies that own many subsidiaries or development sites have a need to be informed of the status of various distributed projects. If an array of displays were strategically located within a compound, each screen could represent one or more of the subsidiaries or distributed development centers. Each of these sites could have control over a screen and send status updates in the form of images and/or movies. As time progresses, a gradual history of the development process would unfold on each screen since a history of commands would be saved. Furthermore, each shot would contain compelling and relevant information. In addition, each site could potentially house an array. In this case, each site can communicate with every other site with the press of a single button and at virtually the same instant. All sites would be up to date. Alternatively, sites that need only update specific other sites can do so by selecting its target screen representative at a remote site.

With regards to company propaganda, the system can be used to constantly cycle through a sequence of text, images, and video that convey information about a company, about its products, and about its superiority to competitors. One could envision displays set up in strategic locations within a company where visitors could be tastefully assaulted by the company's image. The power behind using this system is that this sequence of propaganda can be augmented simply through a user request via an input client to add more cycles or change the cycle entirely. This is achieved through an idle system history displaying mechanism.

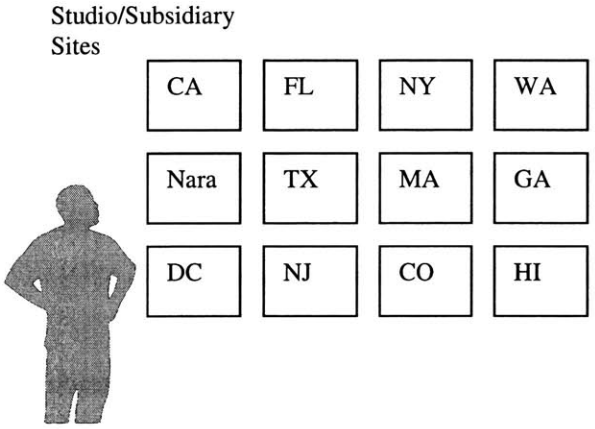


Figure 7.2: Visualization of a development site's status update media sharing wall.

In addition to in-house company propaganda, these systems could be used for visual advertisement on street corners, buildings, blimps that fly over professional and collegiate football stadiums, Times Square, Tokyo, and elsewhere. These areas, especially the latter, are possible locations where masses of people travel every day. A company having the ability to post its propaganda at these junctures bodes well for company visibility and easy marketing. Furthermore, advertisement companies could simply purchase a network of displays situated in various locations, install the system, and offer their services to companies for a fee. In this way, the advertisement company has power over which companies have access to the entire system or specific system nodes located in strategic locations.

Many of these examples can be used in other arenas such as military or intelligence reconnaissance. For the latter, one could envision agents dispatched to various locations and equipped with media recording devices among other things. Each agent can continuously send information back to a home base in real time for immediate data storage and or display.

7.1.2 Recreation

If the general population uses the system, the idea of media messaging becomes a recreational type activity much like AOL Instant Messenger and others have made interactive text messaging recreational. The difference is this system does not discriminate among media types used to message. However unlike these messengers, this system is a more dedicated affair thus sacrificing the aforementioned technologies' simplicity for this technology's media independency.

One could envision a dedicated display intended to receive instructions from the CCS. It is possible that both the CCS and the display client run on the same machine. In this scenario, the display could be used as a centerpiece in a home to receive media messages from family or friends of that household. For instance, a son may have had the opportunity to backpack across Europe. Armed with a digital camera, laptop, or a simple smart phone, this individual could send images or video of his adventures back home. The family could vicariously experience the same trip. Since the display is a dedicated centerpiece, it functions as a constantly changing painting that uses a high-resolution display to depict digital content. The distance gap is essentially zeroed and experiences can be shared from a person to family and friends.

It is also possible to envision games being developed that can exploit the multi-display aspect of the system. Laser tag is fairly popular and arcades tend to have a dedicated area for this type of game. A slightly augmented version of this game could require players to navigate the course as a lone warrior. Display clients can be distributed throughout the

course that provide information on location specific missions and controlled either by automation or a manual worker at the arcade.

7.1.3 Education

In the academic realm, learning by visual stimulation is increasingly being used as the technologies for effectively utilizing such media have improved. Much like the usage described in the business section, systems in this realm can be used in lectures by teachers and professors to display short digital clips that pertain to the topic at hand. Furthermore, students can use the system when presenting a project or an assignment to the class. The system could also be used as a means to teach networking since the system can be broken into explicit parts and tinkered with in real time. It would be a very hands-on approach where the result can be seen almost instantly by students trying to grasp the concept. Furthermore, it can be used to teach new words to young children if used as a giant flash card. A teacher can queue up a lesson plan for the day, connect a dedicated display client to a projector, and project the words of the day onto a screen or wall. As the history mechanism kicks in, a flash card like interface will be observed. Individuals studying on their own could setup a similar environment in their rooms (minus the projector) and cycle through key terms and concepts interspersed with images and video for class specific material.

7.1.4 Art Medium

Using the system as a means of artistic expression is intuitively obvious as a possible function. A museum, school, or artistic symposium can setup a dedicated grid of displays to be used as a digital canvas on which users can paint their creations. A museum could conceivably hold a competition for most creative use of a 4x4 digital canvas for artistic expression and then use the winning entry as an exhibit. A school could purchase a sizeable digital canvas and place it in a location that receives a lot of traffic. Students could then use the canvas for displaying their very own digital creations or sequence masterpieces. Alternatively, the school could start a program that receives digital canvas

submissions and display them according to their own choosing. Professional artists could get together and create some artistic conference or symposium where this system could be used as a decorative piece welcoming participants and adding to the environment. It could also stand as an interactive piece for participants in the event. The most promising use I believe is as a dedicated evolving painting. Professional digital artists could create stunning work. A dedicated centerpiece for display of collected work in one's home also serves as a great aesthetic improvement within a household.

7.2 Extensions

This section will present extensions to the system that have been developed in full or in part. Some of these fulfill usage scenarios described in the previous section.

7.2.1 Architecture for Platform Independent Parallel Computation

This extension has been concurrently in development with the scalable spatially aware media sharing display system. Many of its components have been discussed in the body of this thesis already since these components have significant contributions to the extensibility of the system. The complete system itself remains under construction. In its current state, it functions as a proof of concept for the specific application of thumbnail generation. Although many of the mechanisms are in place, the final piece of that realization is yet to be completed. The mechanisms, such as the Red Black Load Tree and the Parallel Client distinction, provide great springboards for generalization to arbitrary parallel tasks. In addition, the deliberate design of the GeneralComm to pass objects lends more to the possibility of that effort. More on the realization of this functionality is described in Chapter 8. See Appendix A for current progress on this system. The current implementation, including yet to be completed final components, operates as follows:

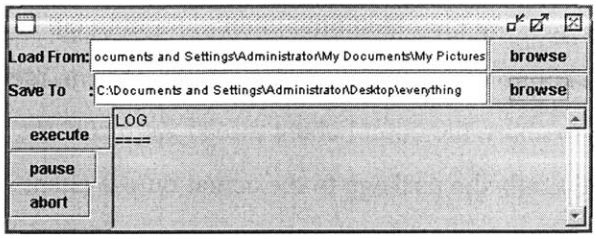


Figure 7.3: Early Parallel Client Application Interface

In order to submit the job to the system, a user loads up the input client program. The user selects a directory containing images needing thumbnails to be generated. Next, the user selects a directory where the thumbnails will be saved. Finally, the user selects execute and may go about his business elsewhere.

From here, the input client program encapsulates all directory information with a set recursion depth (easily adjustable) including directory names, filenames, sizes, and document count. Furthermore identification numbers are assigned to each directory and form nodes on a tree. Next, a compressed version of this tree is built encapsulating all of this data with simple integers and various length encoding schemes.

Once these structures are created, the input client makes a connection request to the CCS. The CCS accepts this request and hands off the communication link to an appropriate client thread. The input client wraps the compressed directory information along with other instructions into a package and sends it off to the CCS. The client thread handling the input client validates this package, and adds the instructions to the CCS command queue.

Once the CCS retrieves the command entry from the queue, it is recognized as a parallel job task and is added to the parallel job queue. Another thread that checks this queue receives the data and extracts the compressed load information including the requesting client's host name. This thread then uses the size information in the tree to adequately balance the load among the available output nodes abstractly represented by the output client threads.

Once the load balancing completes, the commands are sent to each output client thread that handle communications with specific output clients. Again, identifying these packages as parallel job tasks, the output client thread augments the package with some status information and sends the package to the actual output client.

An output client receives the message, identifies it as a parallel job, and initiates a new parallel job handler thread. This thread loads its assigned task and requests a direct connection to the input client identified by the received host name. The input client accepts this connection and receives its first file request information from the output client. Using the compressed structure, the output client knows exactly which files to request and the input client knows exactly how to trivially decode this request. The files are then sent to connected output clients that immediately begin generating thumbnails.

As time passes by, output clients offer up status information to the output threads located on the CCS. They in turn update the output client load status. Output clients send the thumbnails back to the input client in packages as they see fit. Once the job completes, the connections are severed and the user can then and only then submit another job (unless he or she aborted the current job).

7.2.2 Treehouse Studio

Treehouse Studio, a project underway in the Physical Language Workshop within the MIT Media Laboratory, is an open source initiative to bring diversity in creative expression to broad audiences and to facilitate a creative digital economy. The project involves full web deployment, platform independence, and collaborative data sharing. It is being created as a singular infrastructure for gathering, manipulating, and exporting data to and from various sources in a manner exploiting static and mobile connectivity. Within this framework, the media sharing system became a significant component.

Since Treehouse Studio is specifically designed as a collaborative community oriented space for digital content creation, the need to display such content is prevalent. Though

content exists on the web and is accessible anywhere, development has been concurrently underway to allow physical realm outlets for the data. The media sharing display mechanism functions as a great tool to display user work in that realm and was a first proof of concept that such possibilities existed. The user interface in Treehouse Studio for posting content to the area is extremely simple and utilizes JSP technology in the background to handle connections. This user interface is, in essence, an input client to the system (see Figure 7.4).

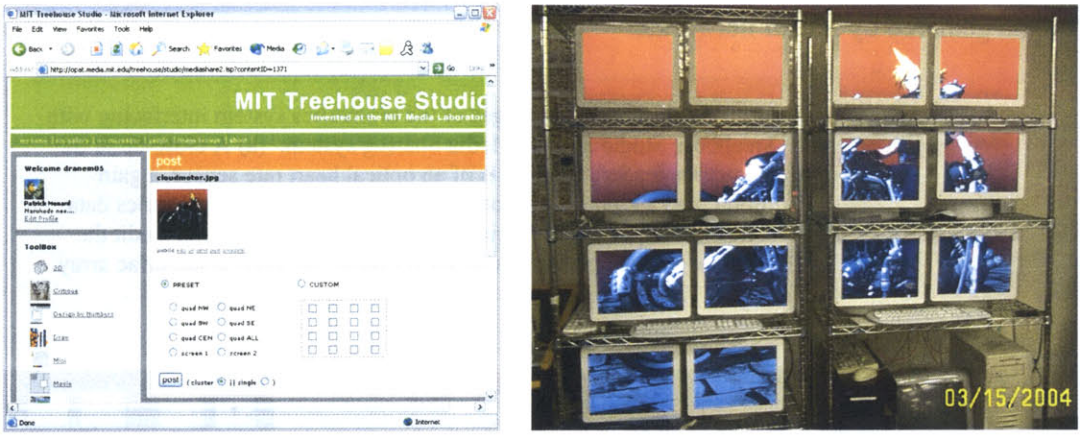


Figure 7.4: Treehouse Studio input client control interface used to post media to the iNode array.

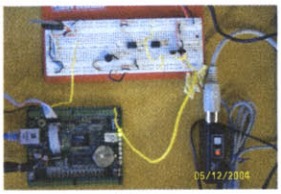
The ability of the system to accept content via email and store them to a repository also became crucial to the Treehouse Studio as a conduit for inserting data into the system outside of the actual website. This ability essentially gave Treehouse Studio a device independent interface for inserting data into the architecture. The media sharing system was augmented to be able to use a simple protocol for contacting the Treehouse Studio database and inserting content for specific users of the system identified by their E-mail addresses.

7.2.3 Biometrics Mail Trickle

The biometrics mail trickle project was an attempt at gathering data from the physical world collected by a sensor and inserting it into the Treehouse Studio (see Section 7.2.2) architecture where other onsite applications can be used to operate on the data. By

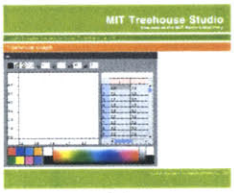
utilizing the E-mail conduit already established by the media sharing system, this type of functionality was realizable. A mechanism for creating images from random data was created as well.

This project opens the door for arbitrary forms of sensor networks to be used for gathering data and channeling it through the media sharing system. A variety of input clients can be created as long as the protocol is obeyed. In the same vein, a variety of output clients with alternate types of functionality can be developed that can operate on the data gathered by sensor networks.



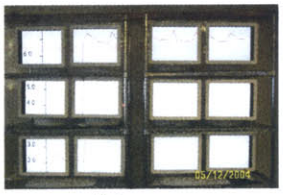
(a)

(b)



(c)

(d)



(e)

Figure 7.5: These are images of the biometrics system interfacing with Treehouse Studio and the Media Sharing system. (a) A close-up of the biometrics reader. It consists of an optical heart rate sensor, a gain circuit, and a rabbit microcontroller. (b) Initiating the biometrics data collection sequence. (c) Using Treehouse Studio to load and edit the data into a visual presentation. (d,e) Posting the data to the iMac array.

8 Evaluation

In this chapter, I will provide an evaluation of the system with regards to contribution, viability, shortcomings and suggestions on improvements. These will be discussed in the sections on comparisons, analysis, and future work.

8.1 Comparisons

Chapter 2 touched on precedents and related work to give a context for my system. In many ways my system is similar in goal and usage to those aforementioned precedents and related mechanisms. The system is similar in its function as a medium for communication and data sharing via varied media types for relevant and enhanced contextual experiences. It contributes and deviates from them in the following ways.

In regards to the precedents, my system exists as a compliment or potential extension of functionality. As discussed, television is used for many purposes including news and recreation. In much the same way, my system can perform these tasks without the need of a camera crew or producer. Operating via the Internet, not only does it act as an extension to the use of the World Wide Web, but it also provides an easy way for different parties to keep each other up to date on important communications (see Chapter 7). Furthermore, it acts as a medium to bring friends and families together even if they are in disparate locations of the world quickly and efficiently (see Chapter 7). Third, my system can function as a tool for purely aesthetic appreciation as a constantly changing painting or digital picture centerpiece (see Chapter 7). The system also acts as an extension to the interactive entertainment realm in principle (the system by definition is interactive). In addition, it is possible to conceive of input clients to the system created specifically with gaming in mind.

In regards to the related work, my system shares many aspects of their design and functionality and also harbors unique contributions. In some form or in no form, much of the related work excluding the thumbnail generators aim to simplify communication

among various components and make life easier for the user by making some context of media or device independence. In these aims, my system is no different. The contributions my system makes are in other forms. My system, in design, seeks to achieve multi-interface control. In other words, it seeks not to limit the number and kinds of devices that can control the mechanism or use the infrastructure. In this endeavor, potentially any network accessible device can operate the architecture. My system also provides the opportunity for remotely sharable display spaces outside of a local area context. Chapter 7 presents many scenarios where such ability is desired and effective. Furthermore, this system supports on the fly output client assimilation. Potentially many output clients can connect to the system thereby increasing its functionally with minimal effort. In addition, users can design varied function output clients to suit their own needs. And finally, my system supports user configurable and scalable display types. A user, by simple use of an input client controller, can decide what kind of scaling or configuration he or she sees fit to harness. He or she can also configure a display array to any two-dimensional sized grid conceivable. In other words, a user of my system has the freedom of flexibility at his or her fingertips.

Apple's Xgrid system and my system have been concurrently in development and came as a shock when they announced such a system. The Xgrid's structure is eerily similar to my own. In addition, both seek to accommodate loosely coupled parallel tasks and automatic node assimilation to boost the computing power of a dynamic parallel cluster. Furthermore, both systems are using commodity components to accomplish this feat. Where the two deviate is primary intent. The Xgrid is primarily a dynamically created grid-computing infrastructure for arbitrary parallel tasks that are loosely coupled. My system is primarily a platform and device independent infrastructure for sharing and/or displaying various media types onto a grid of or singularly placed target display nodes while accommodating dynamic growth.

Work on the parallel extension of the system incorporates more contributions and deviations. The system deviates from the slew of thumbnail generators presented in Chapter 2 in its intent for both single *and* multi-user access and thumbnail generation.

Single users can use it to submit exceptionally large jobs to a parallel processing center for speedup; however the power comes when there exists a body of individuals working toward a unified end, such as a publishing company, a magazine company, or a news company. For these companies, a large body of images is captured daily and so require applications having the batch processing power of the Perl thumbnail generator and the time saving efficiency of parallel operation. My system will allow such companies (and other interested users) to create an in-house "super-computer" infrastructure out of commodity components for bulk image processing (compression in this case). Furthermore, my system's potential computing power increases with the ability to be viral, i.e. turning available computers into potential output clients.

8.2 Analysis

In order to be viable, the system needs to be simple, efficient, sustainable and extensible. All of these requirements have been achieved. The communications framework is fast enough to human perception and use. There are no perceivable delays for conventional use of the system. Conventional use is defined as using an ordinary input client application (such as the Treehouse Studio interface) and displaying simple media such as text and images. Once users begin to send large images, download delays begin to impact the system as far as display is concerned. The QuickTime engine on the display clients take a URL as input and attempts to locate the media on the network. As a result, large media will require time to download. This download time is dependent on network traffic locally as well as to the specific media. In addition this download time becomes dependent on the actual physical output client machine. Movies take time to load since QuickTime must load several internal mechanisms to launch a movie. In addition, the current implementation of the system retrieves the movie and saves it locally before submitting it to the QuickTime rendering engine. The QuickTime mechanisms for downloading a movie from the Internet and loading it in one call are, in high likelihood, more efficient than doing so myself. However a problem with a QuickTime movie player displaying in a java frame and refreshing its display necessitated the workaround.

In a prior version of the system, a busy-wait model was used in the communication framework for retrieving data and adding validated commands to the message queues. A busy-wait was also being used for the observers. This resulted in wasted computation cycles as well as bottlenecks in those areas of the system. In comparison to the notify-wait model, the hit to performance was enormous. After changing to the notify-wait model, the message handling mechanism no longer bogged down the system and escaped bottleneck status.

In order for the control center server to be self-maintaining, deliberate measures of resource allocation and memory management were enacted. During idle periods, resource allocation mechanisms are initiated including garbage collection. To ensure minimal memory waste, careful de-referencing of variables occurs especially within the special purpose data structures discussed in Chapter 6. The server also maintains log files detailing specific operations of the system. In case of major error, the events leading up to a serious crash will have been logged and administrators can uncover the source of the problem. Unfortunately, a mechanism for automatic internal restart has yet to be incorporated within the control center server. Such a mechanism would perform a restart once a day signaling total resource reclamation to the java garbage collector.

Output clients also maintain a system of resource management similar to that of the control center server. Observable evidence of its effectiveness include output clients that can run for a couple days without incident or observable decrease in display time or functionality. However, on occasion the QuickTime engine produces an error that is unrecoverable and causes the Java Virtual Machine to exit. This phenomenon is unacceptable for the requirement of output client life preservation. The reason for this phenomenon is yet to be discovered. However, a Perl script has been created to mask this effect. A minimal program, the script can run for days without incident. It observes the process ids of the output client and the output client background. If this process id vanishes, the script issues a restart of the program. The output client background (a separate Java program) effectively masks the crash by creating the illusion that the media

on screen is about to change. The Perl script does not cause excessive load on the machine hence not affecting performance but improving it.

The system was also put to user testing in a real scenario at the week long Summer Design Institute held in New York from July 12, 2004 till July 16, 2004. The system was used during the exhibition part of our workshop in combination with Treehouse Studio. Presented to educators and designers, Treehouse Studio was used as a vehicle for digital content creation in a collaborative onsite setting. This tested the ability of Treehouse Studio and Media Sharing to be transportable and deployable.

Figure 8.1: Images from SDI event. From left to right: interactive instruction, display, critique, and wrap up.

The entire package itself was termed Treehouse Shuttle. All facets of Treehouse Studio from tools to collaboration and communication via the system were heavily tested by exercises we had developed. During exhibition of work, a single Tablet PC, attached to a mobile digital projector, was used as the output client. Participants were then instructed to send work to the display and present their idea to the group. Without the simplicity of the display mechanism afforded by my system, this exhibition process would have been more involved with regards to setting up and getting work to display on a projector. Instead, the system afforded a more streamlined approach that proved worthwhile, effective, and efficient. The ability for the participant to present their work and the rest of the group to critique was magnified. Furthermore, the workshop conductor's job was greatly simplified. The only suggestion given by the group with regards to the system was an ability to post multiple images to a single output client and be displayed as a collage of some sort. In this way, work can be critiqued in comparison to other work.

8.3 Future Work

There are many areas that can be improved, fine tuned, and extended in the system. There are also numerous scenarios where this system can be utilized. Testing the system in as many of these scenarios as possible will prove favorable to augmenting the design and establishing the system as a practical mechanism. To do so would involve securing user groups in these areas and providing them with the software, support, and means of extensively user testing the system in order to collect more data points.

Moving the system to a more dynamic framework as far as configuration and discovery are concerned is a worthwhile endeavor. Currently the system uses specific ports as listening sockets for communication to input clients, output clients, special clients, and the dedicated special purpose CCS control client. If other programs are using these ports, the CCS itself will abort operation and quit. The clients themselves will attempt to communicate to the CCS via their respective ports but will fail. If the system CCS undertakes a mechanism that can dynamically discover open ports to use for communication, the system will become more robust. A problem with using dynamically allocated ports is in notifying client programs of the port number. Alternatively, the system protocols can be changed to all run on port 8080 initially and be redirected, as they are now, to the dynamically allocated sockets.

The system object models can be improved. Currently there are some components that share similar structure and similar inner classes. These inner classes can be extracted and the other classes can be made to subclass a new class holding common functionality. Furthermore, interfaces can be designed to force certain foundational classes to implement specific methods. By doing so, the system can be made more general and modular by having all core components operate on the interfaces eliminating the need to know what kind of specific object is being operated on. The Content Creator class, currently not being used by the system, is a move in that direction. In addition, each tier of the system can, in theory, function as a control center server leading to more sophisticated and intelligent networks within this framework. Algorithmic enhancements

can also be made throughout the code to further increase execution speed. Some of these include iterative tree generation algorithms, further and better message handler and communication framework abstractions, and a more defined architecture including API and interfacing mechanisms to name a few.

In addition to improvements in system object models, the current mechanisms for parallelism must be generalized to operate on arbitrary jobs. Instead of being specific to files, the load trees can be augmented to simply represent numbered tasks and their sizes. In this way, the load balancing routines can be a more powerful tool. Furthermore, a mechanism for identifying custom media being passed within the framework must be implemented. This involves being able to unwrap and run packaged classes. This type of media could easily be masked as a parallel task since that framework must also support embarrassingly parallel jobs. Furthermore, the specific task of thumbnail generation can be extracted out and designed by extending required parts of a base media sharing system to accommodate that parallel task. Finally, augmenting the infrastructure to allow non-Java enabled devices to capitalize on a parallel framework would increase the device and platform independent goal of the system.

While architecting the communication framework, I intended to use GZIP Output and Input Streams encapsulated within the GeneralComm framework to allay the load on network communications. However, the code exhibited extremely odd behavior, simply blocking for an extended time on the input stream end. By all indicators, this event should not have occurred (especially after researching its use among other applications including syntax and structure). However the problem persisted. On the one hand, the actual gzipping process employed by the stream may not be worth the time if simply passing the data as is through the network occurs much faster and for most commands and loads.

Using another method of determining location can further extend the idea of spatially aware output clients. For instance, bluetooth location sensing or GPS can be harnessed. Doing so would introduce real location instead of relative location.

Under the current implementation, a vague sense of user is present in the system. As a result, any person can send requests to the system. For security, a more defined sense of user can be developed allowing administrators to create some form of access control to the system. In addition to this notion of user, a further extension involves allowing deletions from the storage mechanism. If a notion of user exists, such a mechanism will not be as dangerous as it could otherwise.

Video in and of itself introduces another complexity to the system – synchronous feeds. When the display type is selected as a cluster entity using video content, output client displays must be able to coordinate the display such that the video plays synchronously across the desired user array space. An alternative to handling coordination is to receive video from the CCS at multi-cast addresses and play the bytes received as a live feed. This opens the door for more sophisticated control from input clients. Supporting truly streaming data in general functions as a necessary extension in putting the output clients to pertinent use. Streaming data will allow for more creative user applications to emerge taking advantage of the system.

The history mechanism in the system can be augmented to support direct control from the user or the administrator. If explicit or undesired material creeps into the command history, administrators should be able to remove such content, especially if the display is being used in a public area. Users, on the other hand, may request the evolution of a specific piece over time. Since the system supports renaming new files in case of name collision, it could be that the files form an evolution of an idea. This evolution can be sent via email to a user or displayed on the array.

Regarding the issue of custom input clients and sensor networks, one can envision a piece of low level hardware designed to periodically post data to the system, or store some collected data in the system storage repository.

There are yet many other features and enhancements that can amend this system. Adding the ability of output client nodes to communicate with each other in order to share data could enhance parallel tasks or open the door for more interesting display mechanics and control by a user. As far as client programs are concerned, converting to the Java Web Start framework will enhance its portability and platform independent design. Being able target specific output clients for parallel applications has benefits as well. Clustering output client nodes for parallel tasks translates into making location dependent nodes aware to users in a meaningful and streamlined way.

In all of these things, simplicity on the side of the user must be maintained including an infrastructure that supports mobile connectivity.

9 Summary

I have presented an infrastructure of a scalable spatially aware media sharing display system. The system itself is primarily designed to enhance communication among many facets of society. The fact that the main part of the system is centrally located, can be controlled by a myriad of devices, and can integrate output client nodes distributed over large distances provides great opportunities for new and sophisticated applications. The potential effects of controlling these displays could have great repercussions in the various areas of society including business, marketing, research conferences, multimedia events, family and friend interactions, and travel. More importantly, the infrastructure's flexibility and support for mobile connectivity allows any user anywhere to be able to use this system. This means that people who have limited access to useful technologies can have at their fingertips a simple messaging system to connect with people the world over or a limited super computer that uses commodity components. This system promotes the creation of platform and device independent technologies that can unite the many technologies already extant and enhance communication in significant ways. Furthermore, it promotes designing interfaces that can be forwards and backwards compatible and flexible enough to allow motivated users to build more sophisticated control and applications of the system.

References

[1] Apple computer, Inc. "Programming Topic: Rendezvous Network Services," 2003, [Online] Available; http://developer.apple.com/techpubs/macosx/Cocoa/TasksAndConcepts/ProgrammingTopics/NetServices/index.html#//apple_ref/doc/uid/10000119i.

[2] Apple computer, Inc. "Rendezvous Network Services Architecture," 2003, [Online] Available; http://developer.apple.com/techpubs/macosx/Cocoa/TasksAndConcepts/ProgrammingTopics/NetServices/Concepts/NetServicesArchitecutre.html.

[3] "ARPANET," 2004, [Online] Available; http://en.wikipedia.org/wiki/ARPANET.

[4] Ballerstaller, Christian. "The Code Project – Thumbnail Generator," 2003, [Online] Available; http://www.codeproject.com/csharp/thumbgenerator.asp.

[5] Beam, Michael. "Networking in Cocoa," *Mac Devcenter*,2003, [Online] Available; http://www.macdevcenter.com/pub/a/mac/2003/05/13/cocoa.html.

[6] "Blogs," 2004, [Online] Available; http://en.wikipedia.org/wiki/Blogs.

[7] Cappy, "DOOM3 versus Half-Life 2 – HW tema," 2003 (in Czech), [Online] Available; http://games.tiscali.cz/hardware/tema/doom3vshalflife2/.

[8] Cohen, Peter. "InFocus LiteShow brings Wi-Fi to projectors," 2003, [Online] Available; http://maccentral.macworld.com/news/2003/05/22/liteshow/.

[9] "Computer and Video Game Genres," 2004, [Online] Available; http://en.wikipedia.org/wiki/Computer_and_video_game_genres#Educational.

[10] D. McCormack, "Integrating Xgrid into Cocoa Applications, Part 1," *Mac Devcenter*, 11 May 2004, [Online] Available; http://www.macdevcenter.com/pub/a/mac/2004/05/11/xgrid_pt1.html.

[11] D. Rejeski and Ben Sawyer, "Serious Games Initiative," [Online] Available; http://www.seriousgames.org/about.html.

[12] Dauger, Dean E. "Parallel OperatiOn and Control Heuristic Application," June 20, 2003, [Online] Available; http://www.daugerresearch.com/pooch/PoochManual.pdf, http://www.daugerresearch.com/pooch/manual/.

[13] "Extreme Internet Software: digital imaging software for web," 2001-2004, [Online] Available; http://www.exisoftware.com/thumbnail_generator/index.html.

[14] "Extreme Thumbnail Generator Review at Free Downloads Center," 2001-2002, [Online] Available; http://www.freedownloadscenter.com/Reviews/r1080.html.

[15] "Far-Off Speakers Seen as Well as Heard Here in a Test of Television," *New York Times*, 8 Apr. 1927, [Online] Available; http://www.att.com/history/television/nytimes_article.html.

[16] "Games to Teach," 2003, [Online] Available; http://icampus.mit.edu/projects/GamesToTeach.shtml.

[17] Gay, Jonathan and Sarah Allen. "Macromedia Flash Communication Server MX: Use Cases and Feature Overview for Rich Media, Messaging, and Collaboration," July 2002, pp. 1-7, [Online] Available; http://www.macromedia.com/devnet/mx/flashcom/articles/comserver.pdf.

[18] Gluck, Ortwin. "Thumbnail Generator," 2003, [Online] Available; http://www.odi.ch/prog/thumbnail.php.

[19] H. Jenkins, "Game Theory: Digital Renaissance," *Technology Review*, MIT, 29 March, 2002.

[20] H. Jenkins et al, "Games-to-Teach Vision," 2000, [Online] Available; http://www.educationarcade.org/gtt/research.html.

[21] Hanrahan, Pat. "Scalable Graphics using Commodity Graphics Systems," May 2000, [Online] Available; http://graphics.stanford.edu/projects/multigraphics/talks/viewspi.may00/viewspi.may00.pdf.

[22] "History of the Internet," 2004, [Online] Available; http://en.wikipedia.org/wiki/History_of_the_Internet.

[23] "History of Video Game Consoles," 2004, [Online] Available; http://en.wikipedia.org/wiki/History_of_video_game_consoles.

[24] Humphreys, Greg and et al. "Distributed Rendering for Scalable Displays," 2000, [Online] Available; http://graphics.stanford.edu/papers/clust_render/clust_render.pdf.

[25] "Internet Backbone," 2004, [Online] Available; http://en.wikipedia.org/wiki/Internet_backbone.

[26] J. Larson, "What is the Deal with Televised War Coverage?," *American Daily News & Commentaray*, 23 March 2003, [Online] Available; http://www.americandailycom/article/3706.

[27] J. Yu, "The Online Guide to Controversial Video Games," 2002, [Online] Available; http://www.boilingpoint.com/~jasonyu/cs240/.

[28] L. Grossman, "The Age of Doom," *Time Magazine*, 9 Aug. 2004, [Online] Available; http://www.time.com/time/magazine/article/0,9171,1101040809-674778,00.html, http://www.time.com/time/archive/preview/0,10987,1101040809-674778,00.html.

[29] L. Herman et al, "The ultimate History of Video Games," http://www.gamespot.com/gamespot/features/video/hov/p4_02.html.

[30] "Massively Multiplayer Online Game," 2004, [Online] Available; http://en.wikipedia.org/wiki/MMOG.

[31] P. Baran, "I. Introduction to Distributed Communications Network," *On Distrubuted Communications,* tech. memo, RM-3420-PR, Rand Corporation, Santa Monica, 1964, [Online] Available; http://www.rand.org/publications/RM/RM3420/.

[32] P. Baran, "XI. Summary Overview," *On Distrubuted Communications*, tech. memo, RM-3420-PR, Rand Corporation, Santa Monica, 1964, [Online] Available; http://www.rand.org/publications/RM/RM3767.summary.html/.

[33] "Publications in the 'On Distributed Communications' Series," 1994-2004, [Online] Available; http://www.rand.org/publications/RM/baran.list.html.

[34] P. Baran, "I. Introduction to Distributed Communications Network," *On Distrubuted Communications,* tech. memo, RM-3420-PR, Rand Corporation, Santa Monica, 1964, chapter 4, [Online] Available; http://www.rand.org/publications/RM/RM3420/RM3420.chapter4.html.

[35] R. W. Wiggins, "The Effects of September 11 on the Leading Search Engine," *First Monday Peer-Reviewed Journal on the Internet*, vol.7 number 10, 3 October 2001, [Online] Available; http://www.firstmonday.dk/issues/issue6_10/wiggins/.

[36] Skiljan, Irfan. "What is IrfanView?" 2004, [Online]; http://www.irfanview.com.

[37] T. H. Cormen et al, *Introduction to Algorithms*, MIT Press, 2001, pp. 273-293.

[38] "Television," 2004, [Online] Available; http://en.wikipedia.org/wiki/Television.

[39] "Television History of AT&T and Television," 2004, [Online] Available; http://www.att.com/history/television/.

[40] "Television History – The First 75 Years," 2001-2004; http://www.tvhistory.tv/pre-1935.htm.

[41] UCP Morgen. "Media Messaging Solutions," pp. 1-2, [Online] Available; http://www.ucpmorgen.com/downloads/ucpmorgen_media-messaging-solutions_en.pdf.

[42] "Video Game," 2004, [Online] Available; http://en.wikipedia.org/wiki/Video_games.

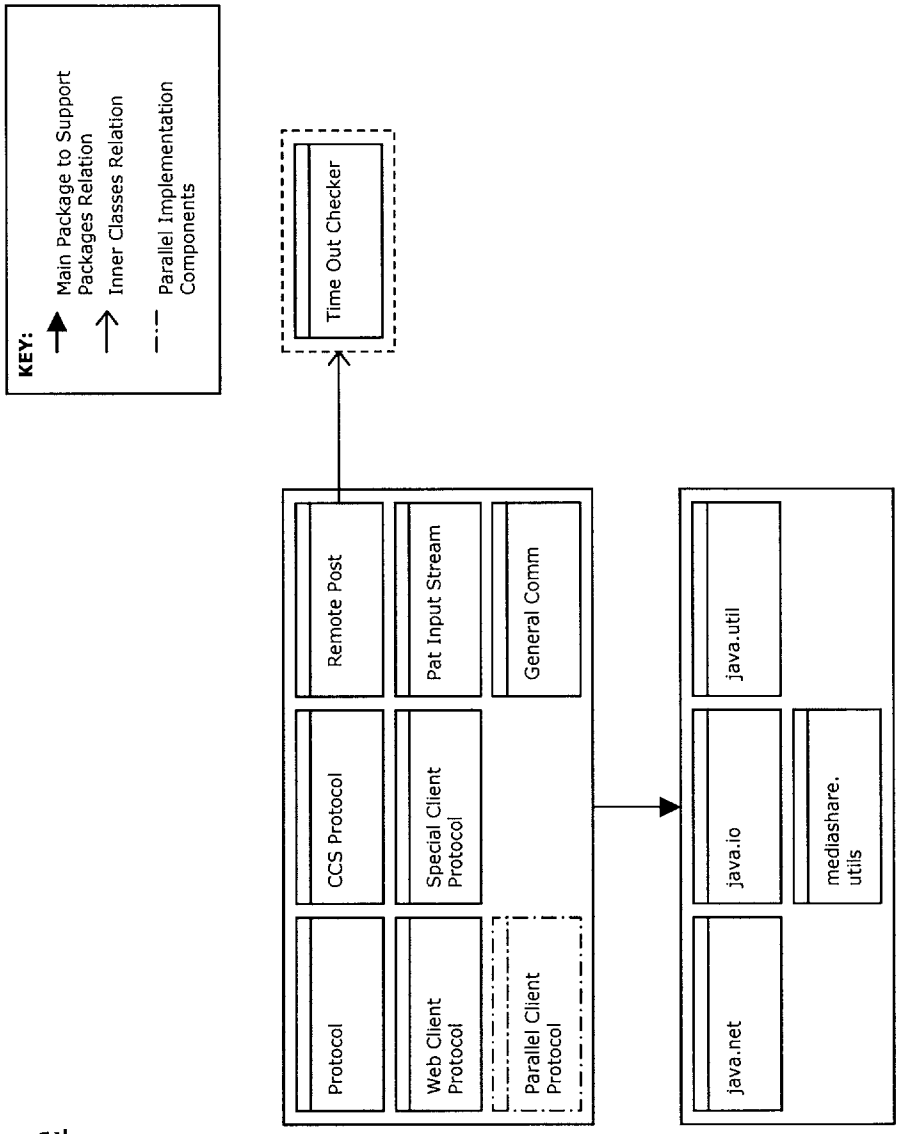
[43] "Wiki," 2004, [Online] Available; http://en.wikipedia.org/wiki/Wiki.

[44] "Xgrid," Advanced Computation Group, 2004, [Online] Available; http://www.apple.com/acg/xgrid/.

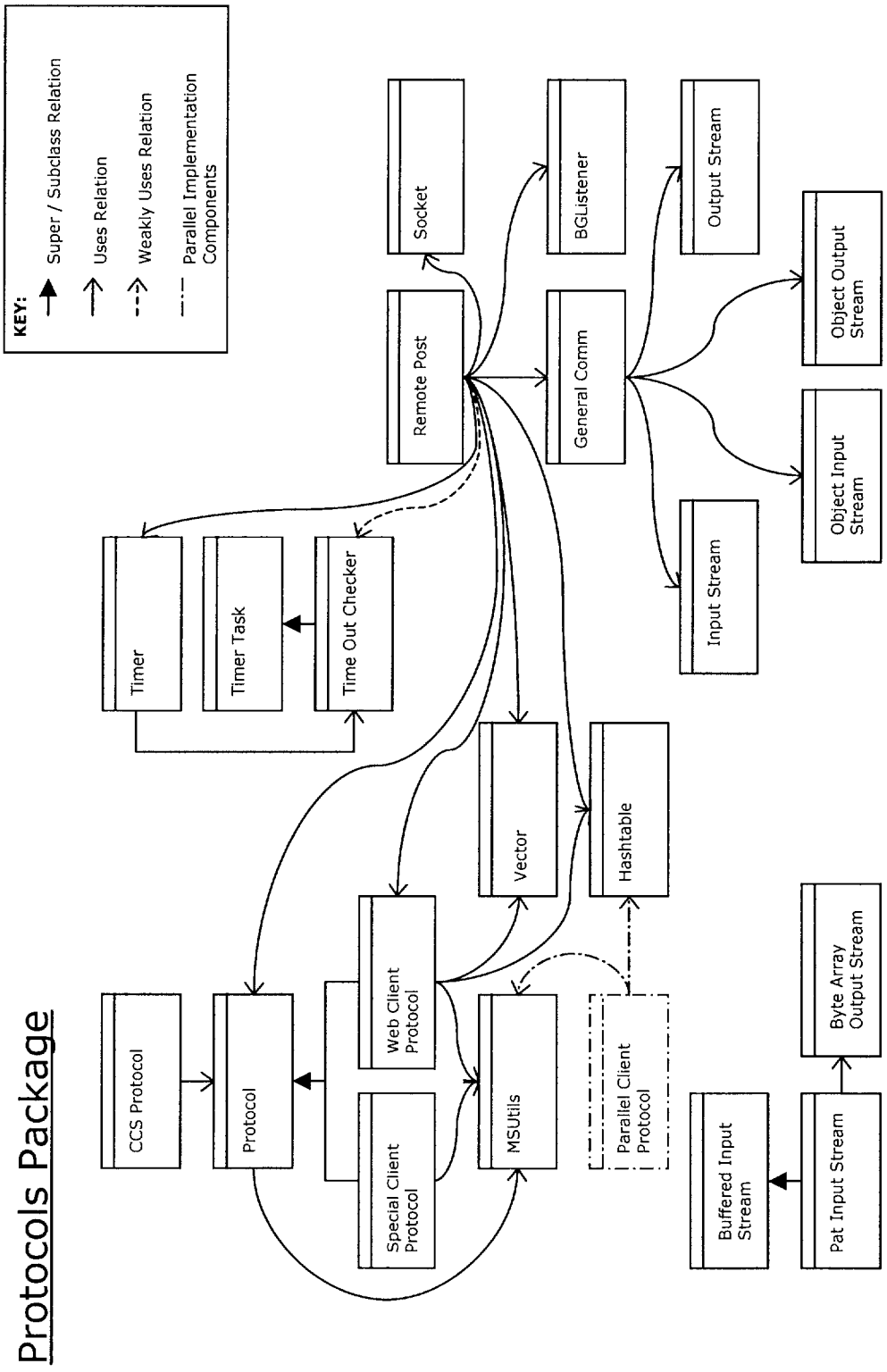
[45] *Xgrid Guide*, Preliminary, Apple Computer, Inc., Cupertino, CA, 2004.

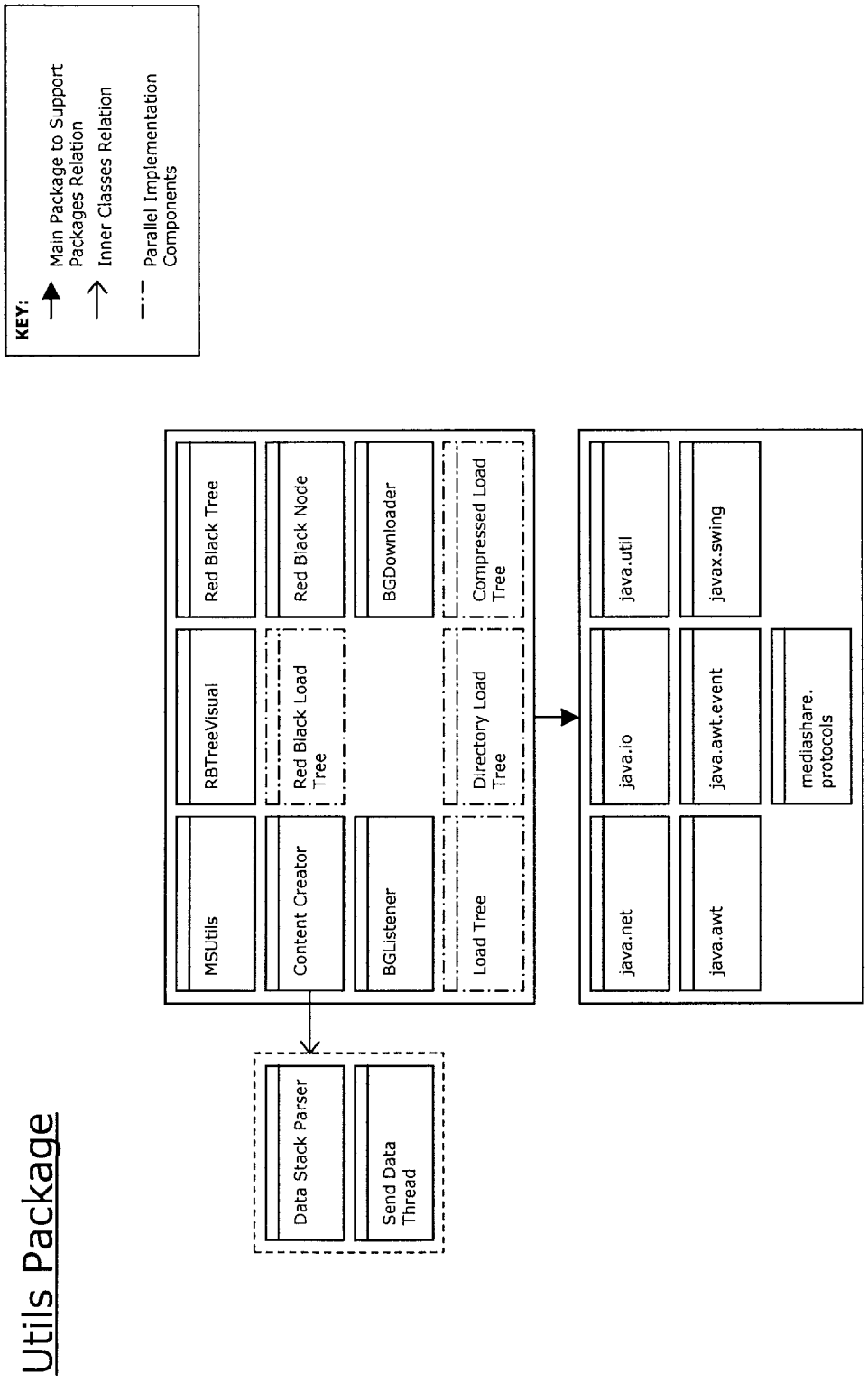
[46] "Ximage Thumbnail Generator," 2004, [Online] Available; http://www.hotscripts.com/Detailed/31815.html.

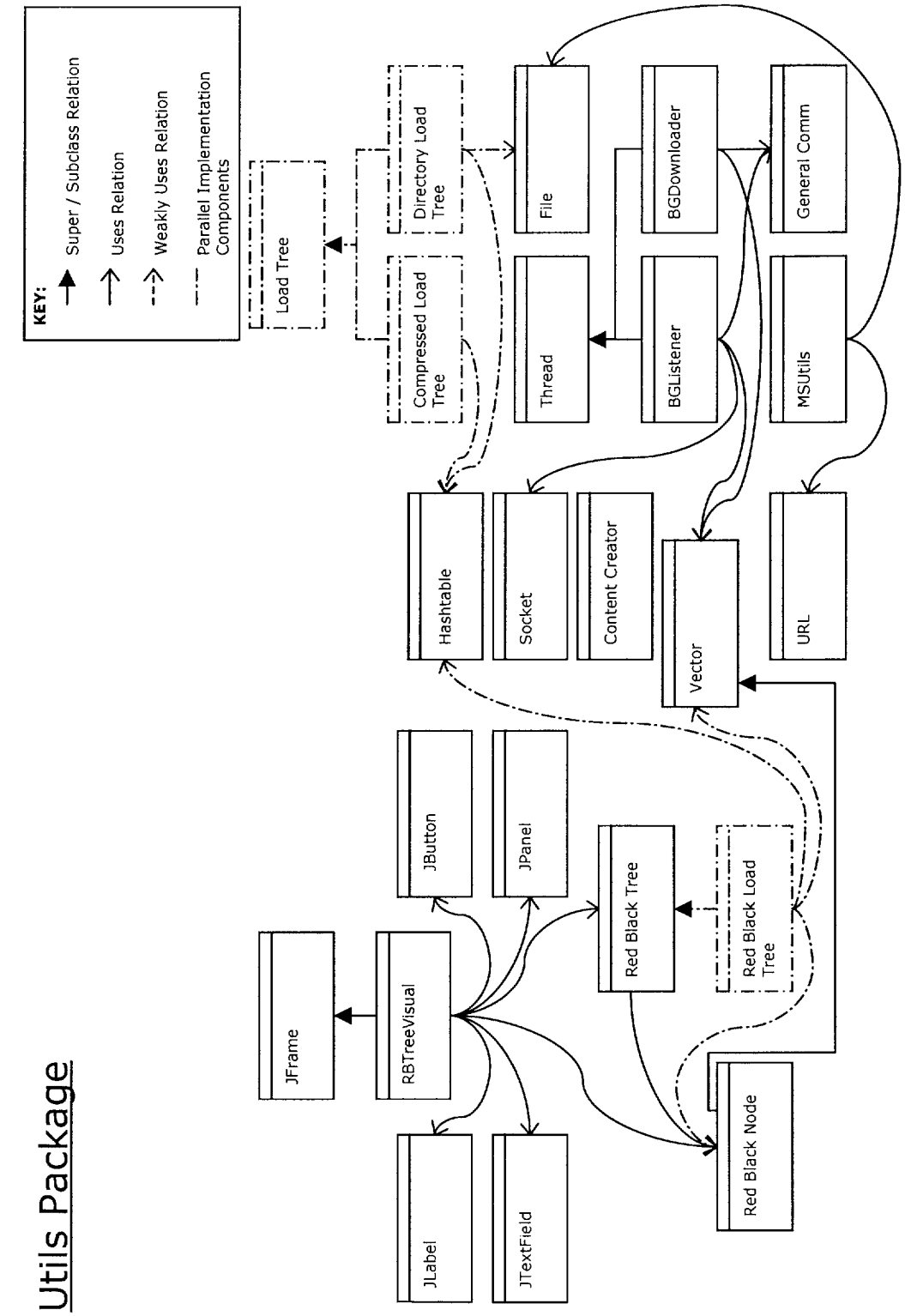
Module Dependency Diagram **Appendix A1**



Protocols Package

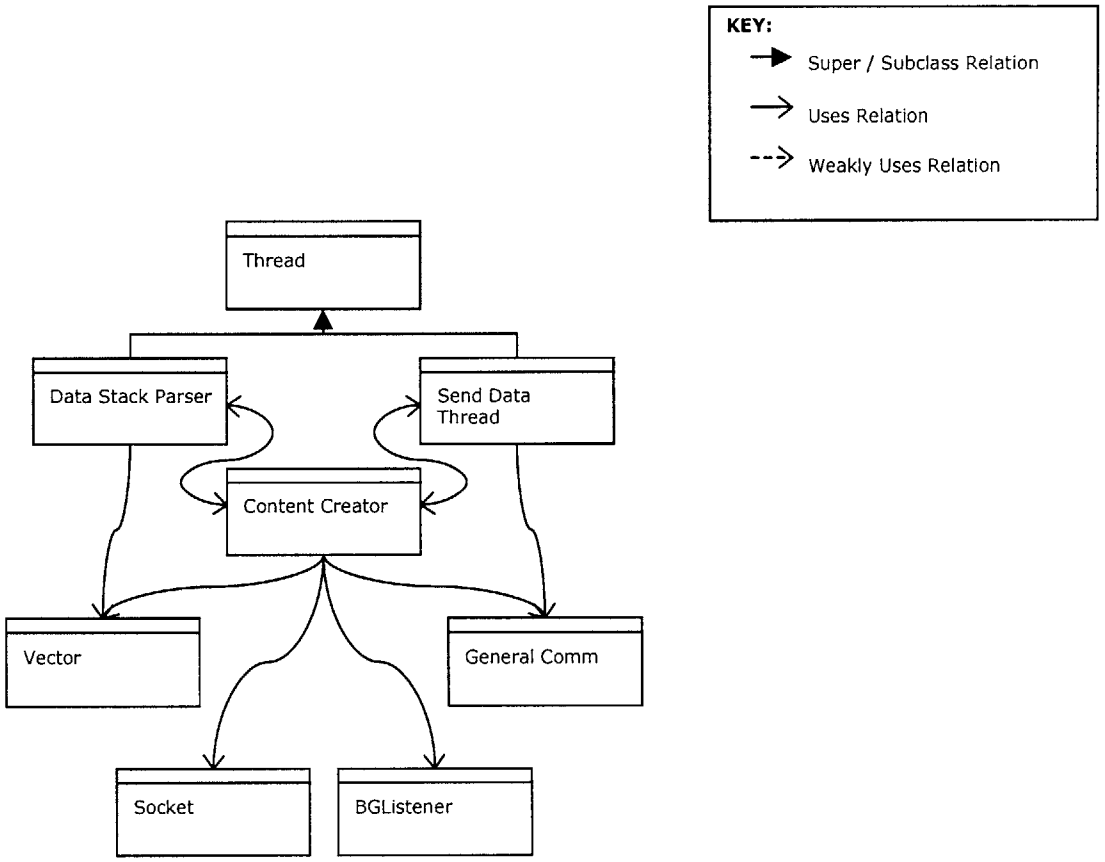

KEY:
Super / Subclass Relation
Uses Relation
Weakly Uses Relation
Parallel Implementation Components
Protocols Package
CCS Protocol
Protocol
Web Client Protocol
Special Client Protocol
MSUtils
Parallel Client Protocol
Vector
Hashtable
Timer
Timer Task
Time Out Checker
Socket
BGListener
Output Stream
Remote Post
General Comm
Object Output Stream
Object Input Stream
Input Stream
Buffered Input Stream
Pat Input Stream
Byte Array Output Stream

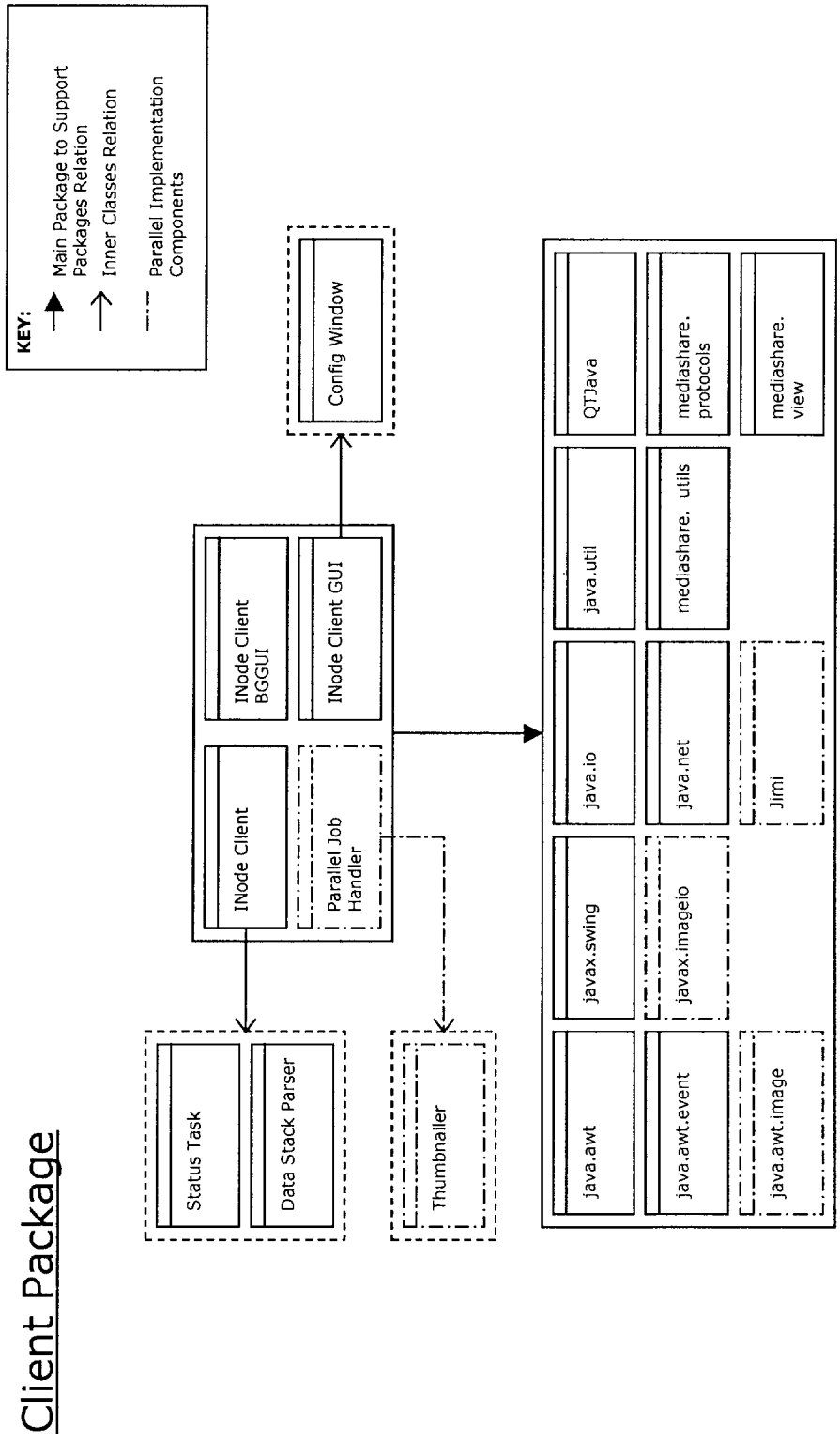

KEY:
Main Package to Support Packages Relation
Inner Classes Relation
Parallel Implementation Components
Utils Package
MSUtils
RBTreeVisual
Red Black Tree
Content Creator
Red Black Load Tree
Red Black Node
BGListener
BGDownloader
Load Tree
Directory Load Tree
Compressed Load Tree
java.net
java.io
java.util
java.awt
java.awt.event
javax.swing
mediashare.protocols
Data Stack Parser
Send Data Thread



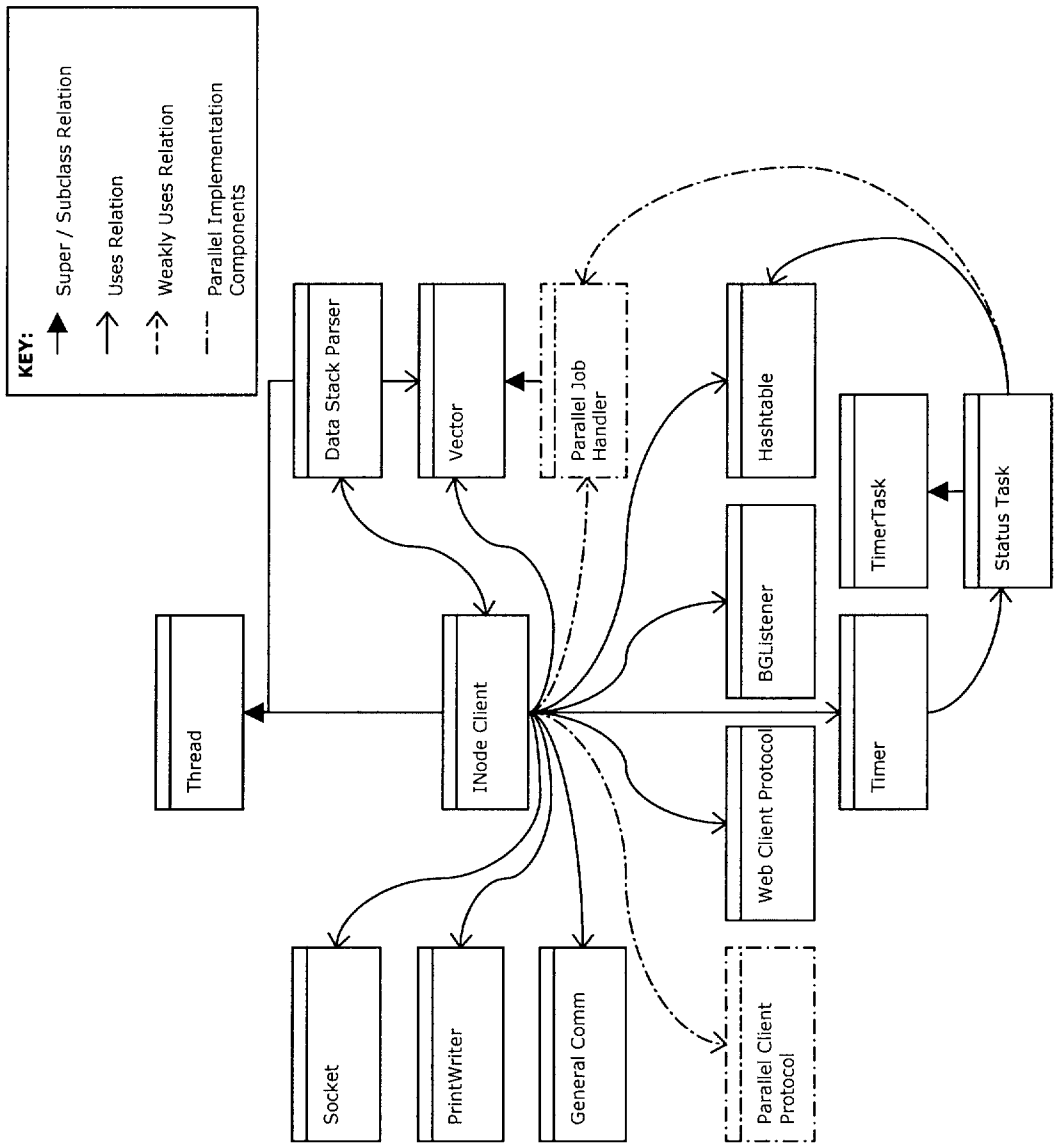
Utils Package

Utils Package




KEY:
Main Package to Support Packages Relation
Inner Classes Relation
Parallel Implementation Components
Client Package
Config Window
INode Client BGGUI
INode Client GUI
INode Client
Parallel Job Handler
Status Task
Data Stack Parser
Thumbnailer
QTJava
java.util
mediashare. protocols
mediashare. view
java.io
mediashare. utils
java.net
Jimi
javax.swing
javax.imageio
java.awt
java.awt.event
java.awt.image

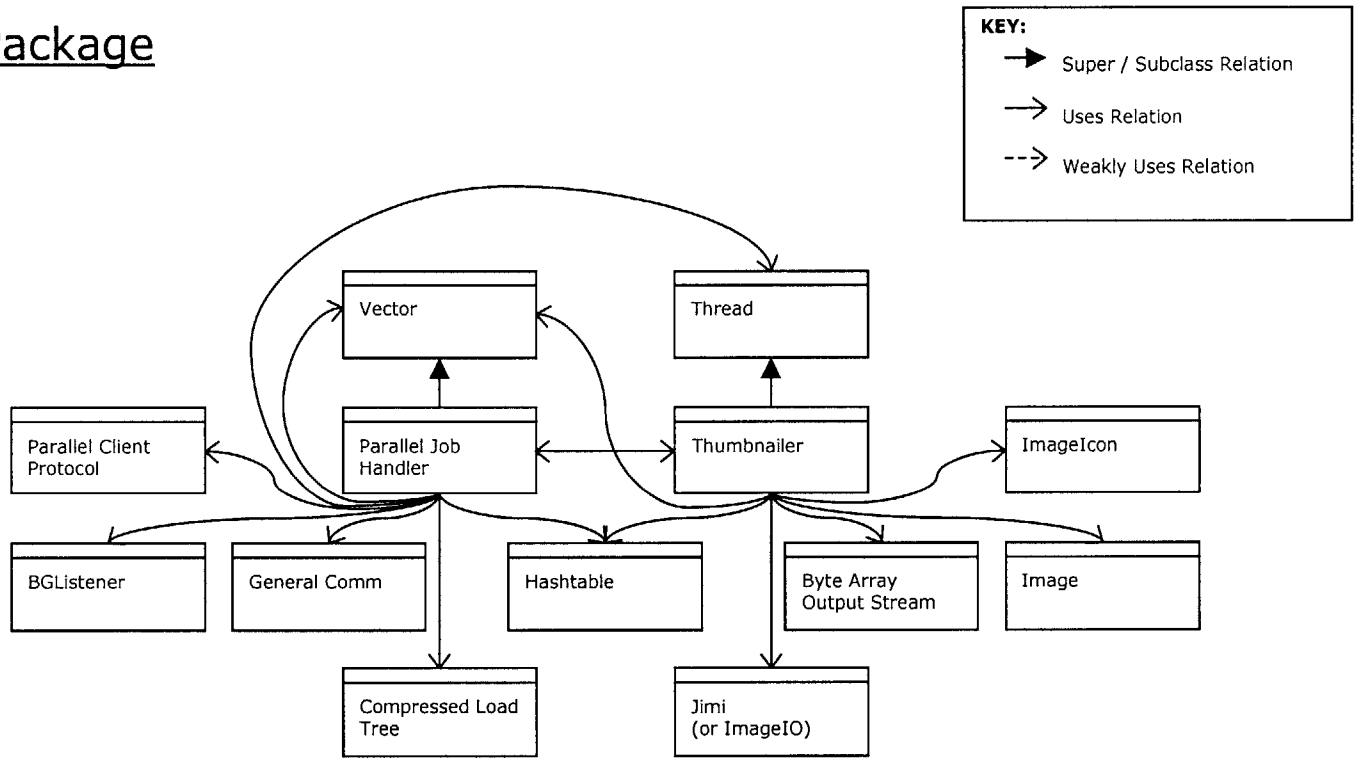
Client Package

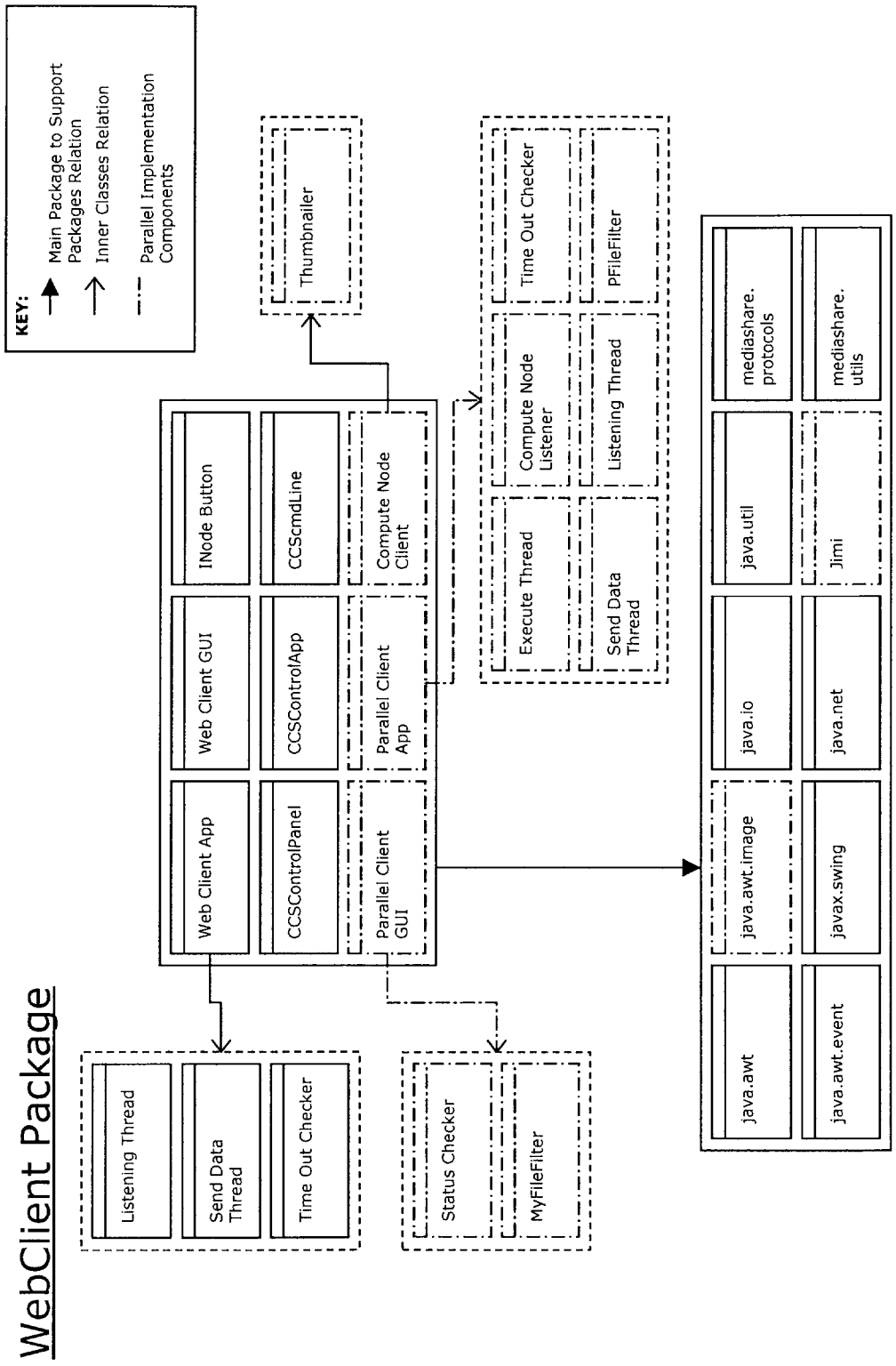


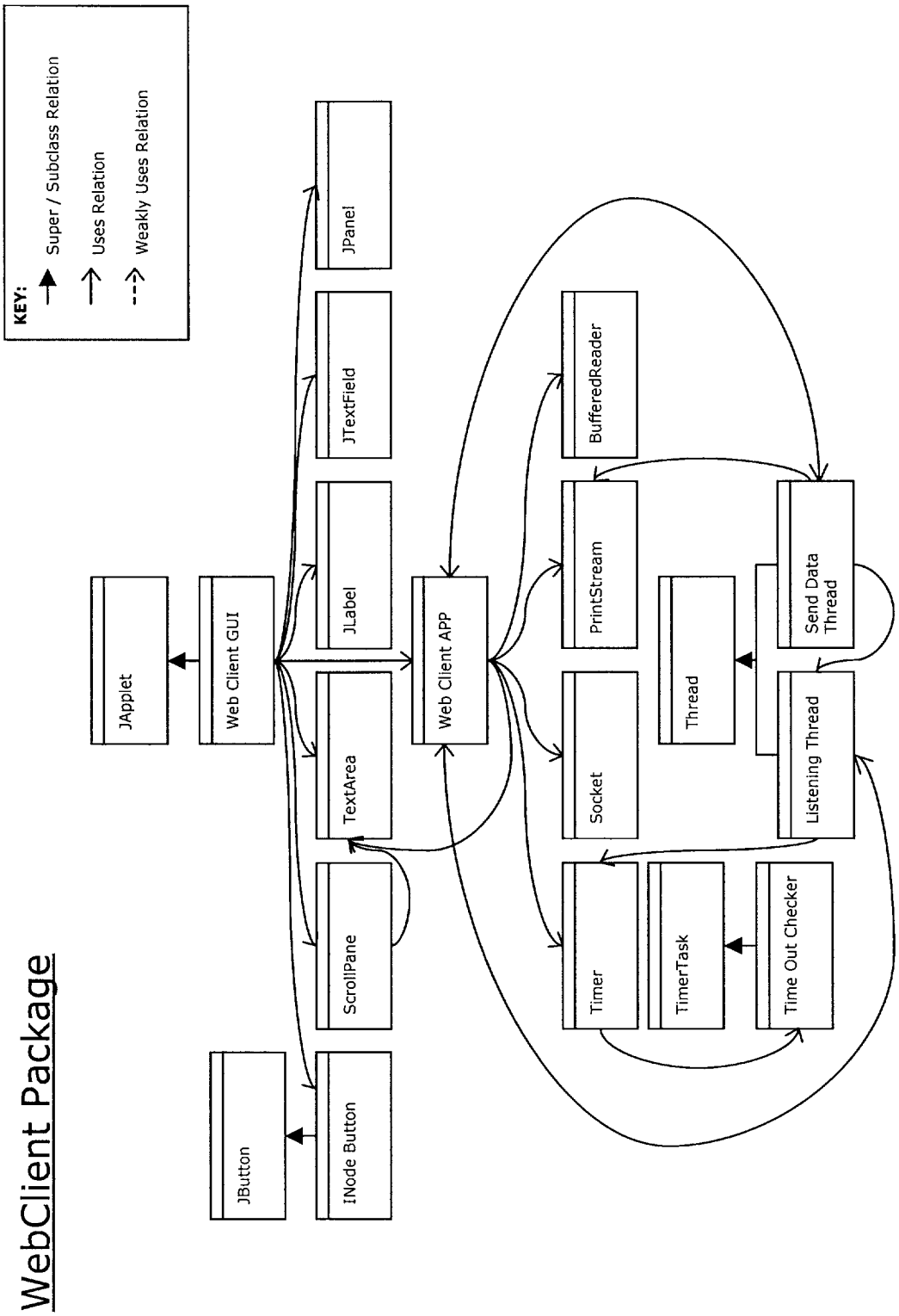
Client Package

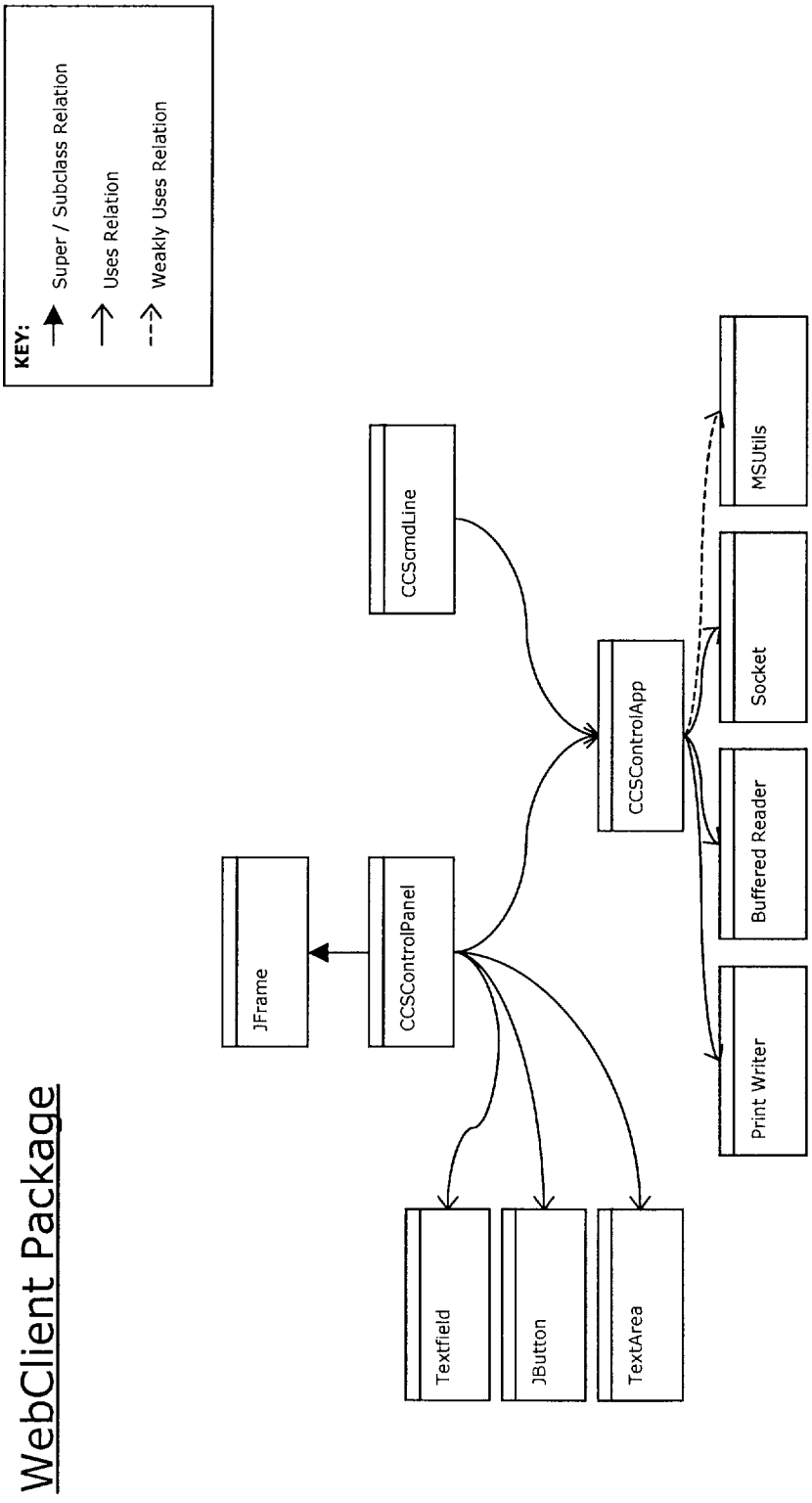
Client Package

Parallel Only



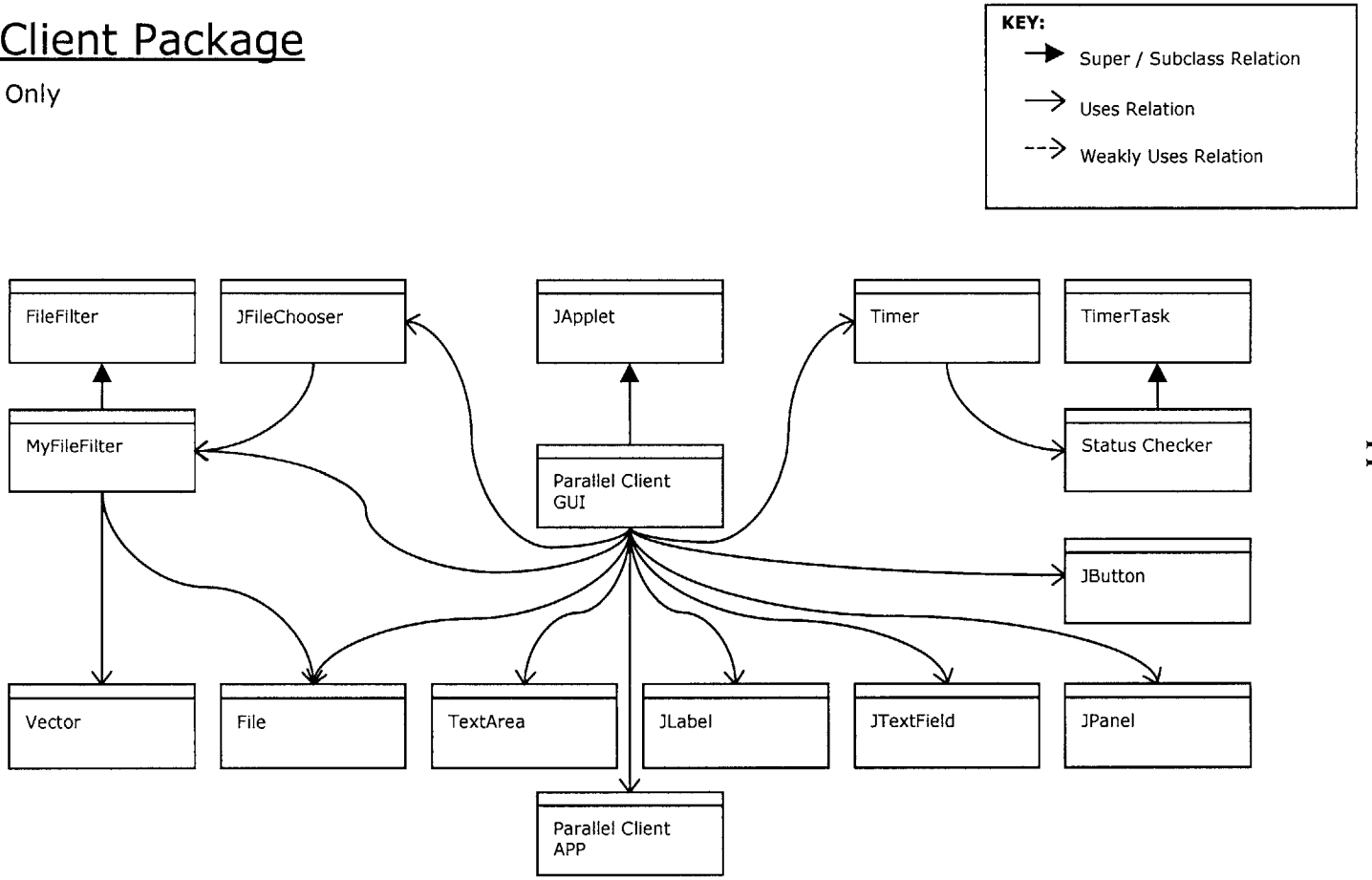

KEY:
Main Package to Support Packages Relation
Inner Classes Relation
Parallel Implementation Components
WebClient Package
Web Client App
Web Client GUI
INode Button
CCSControlPanel
CCSControlApp
CCSCmdLine
Parallel Client GUI
Parallel Client App
Compute Node Client
Thumbnailer
Execute Thread
Compute Node Listener
Time Out Checker
Send Data Thread
Listening Thread
PFileFilter
Listening Thread
Send Data Thread
Time Out Checker
Status Checker
MyFileFilter
java.awt
java.awt.image
java.io
java.util
mediashare. protocols
java.awt.event
javax.swing
java.net
Jimi
mediashare. utils

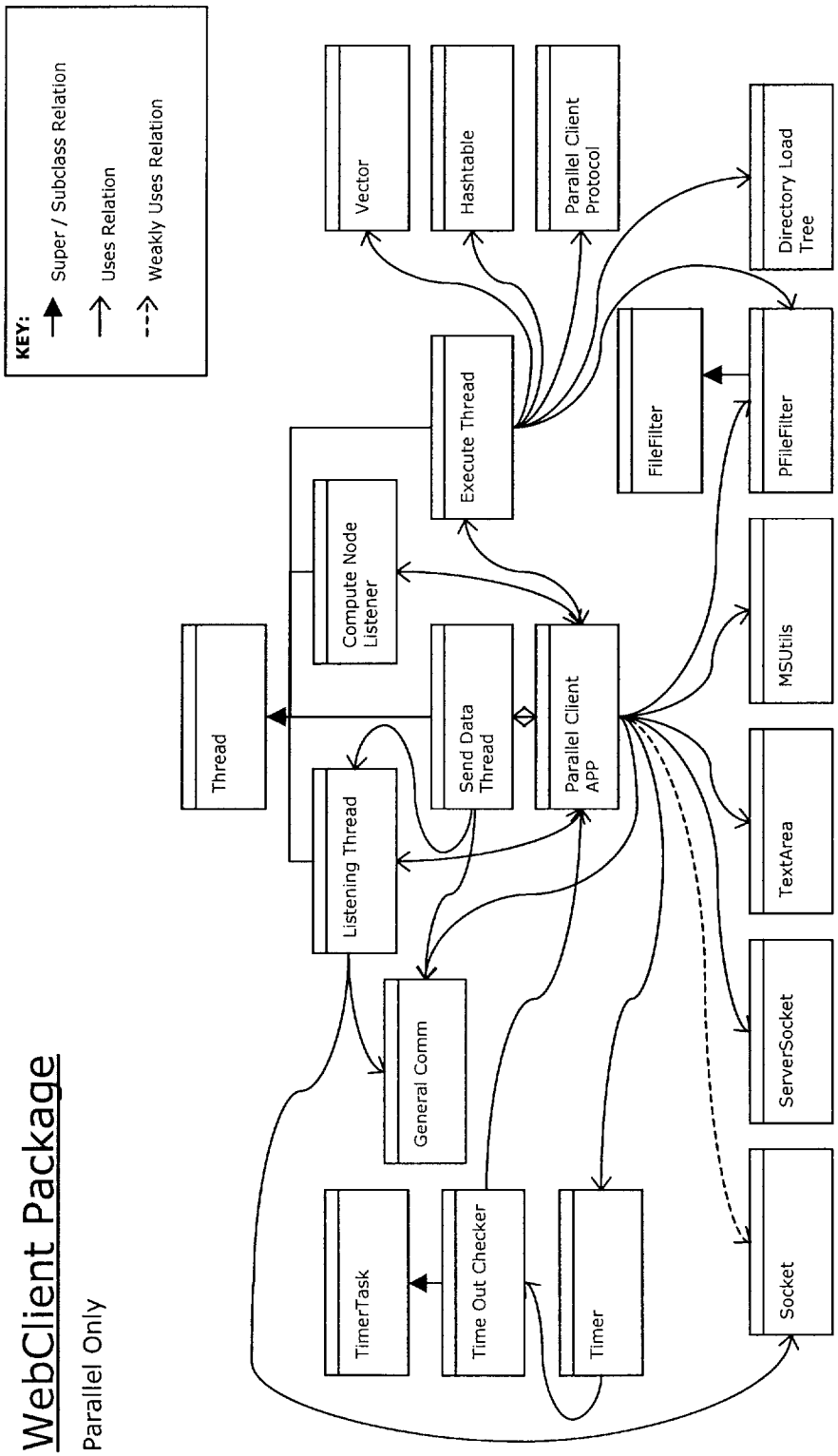

WebClient Package
KEY:
Super / Subclass Relation
Uses Relation
Weakly Uses Relation
JApplet
Web Client GUI
JLabel
JTextField
JPanel
TextArea
ScrollPane
JButton
INode Button
Web Client APP
BufferedReader
PrintStream
Socket
Thread
Send Data Thread
Listening Thread
Timer
TimerTask
Time Out Checker


KEY:
Super / Subclass Relation
Uses Relation
Weakly Uses Relation
WebClient Package
JFrame
CCSControlPanel
CCSCmdLine
CCSControlApp
MSUtils
Socket
Buffered Reader
Print Writer
Textfield
JButton
TextArea

WebClient Package

Parallel Only



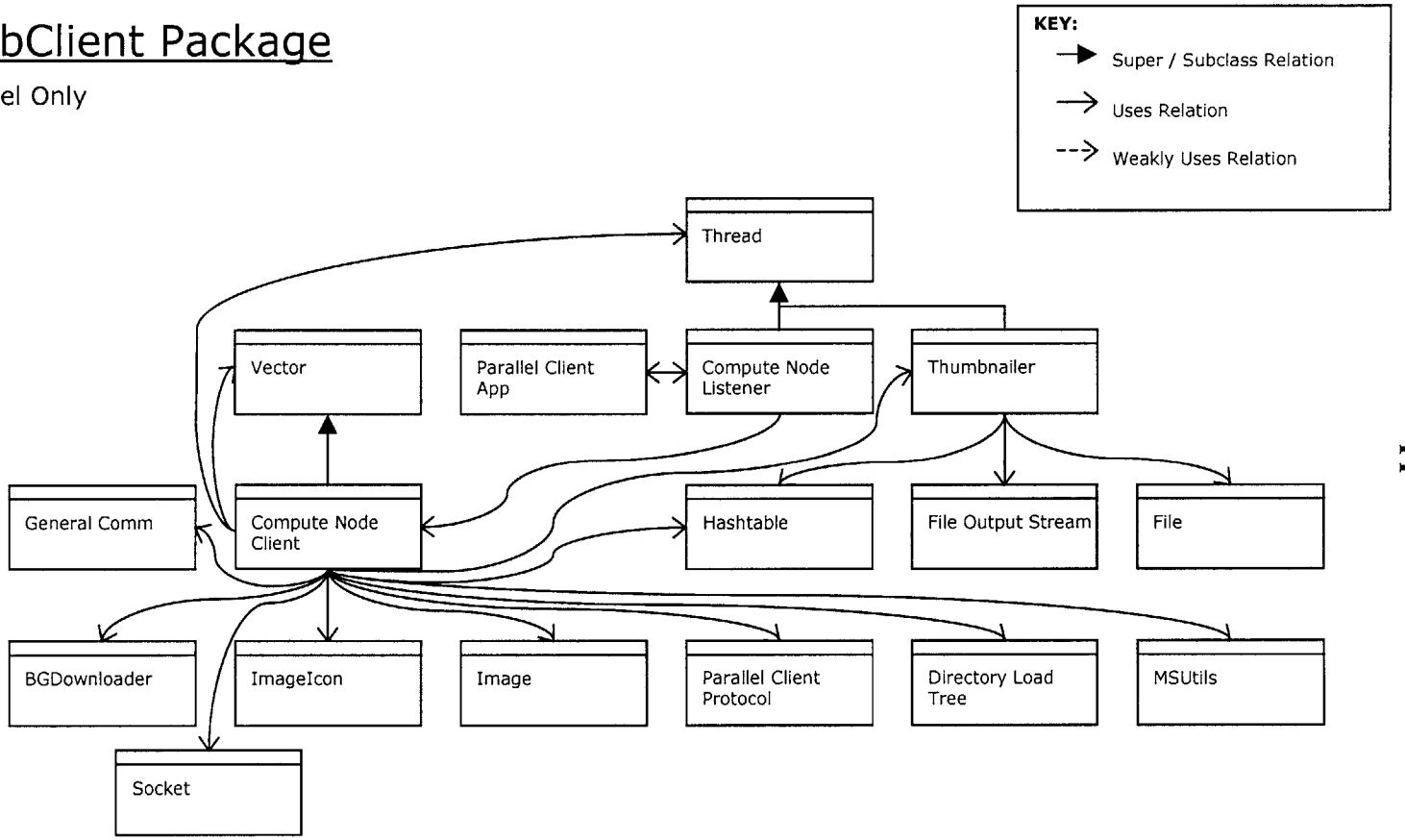


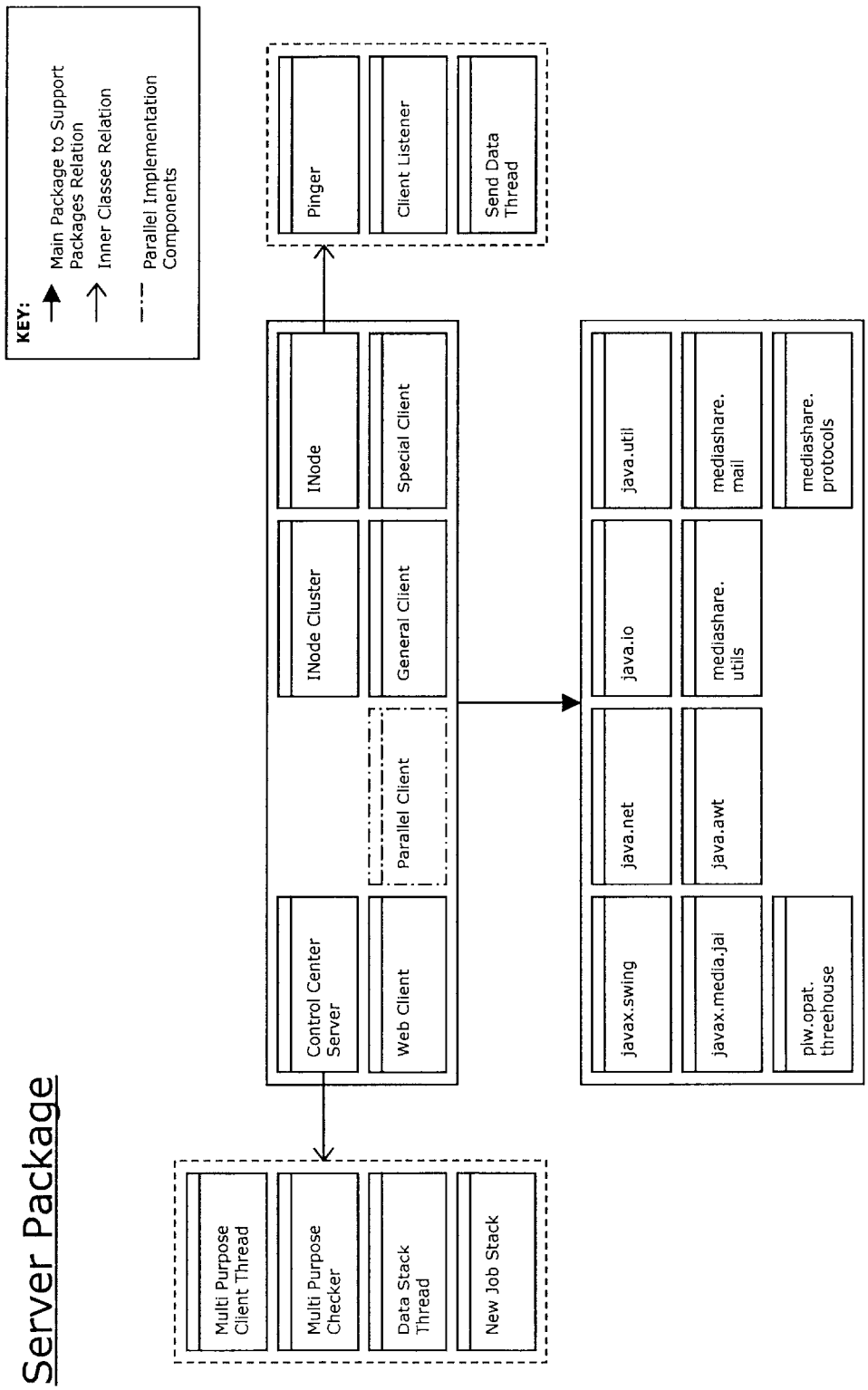
WebClient Package

Parallel Only

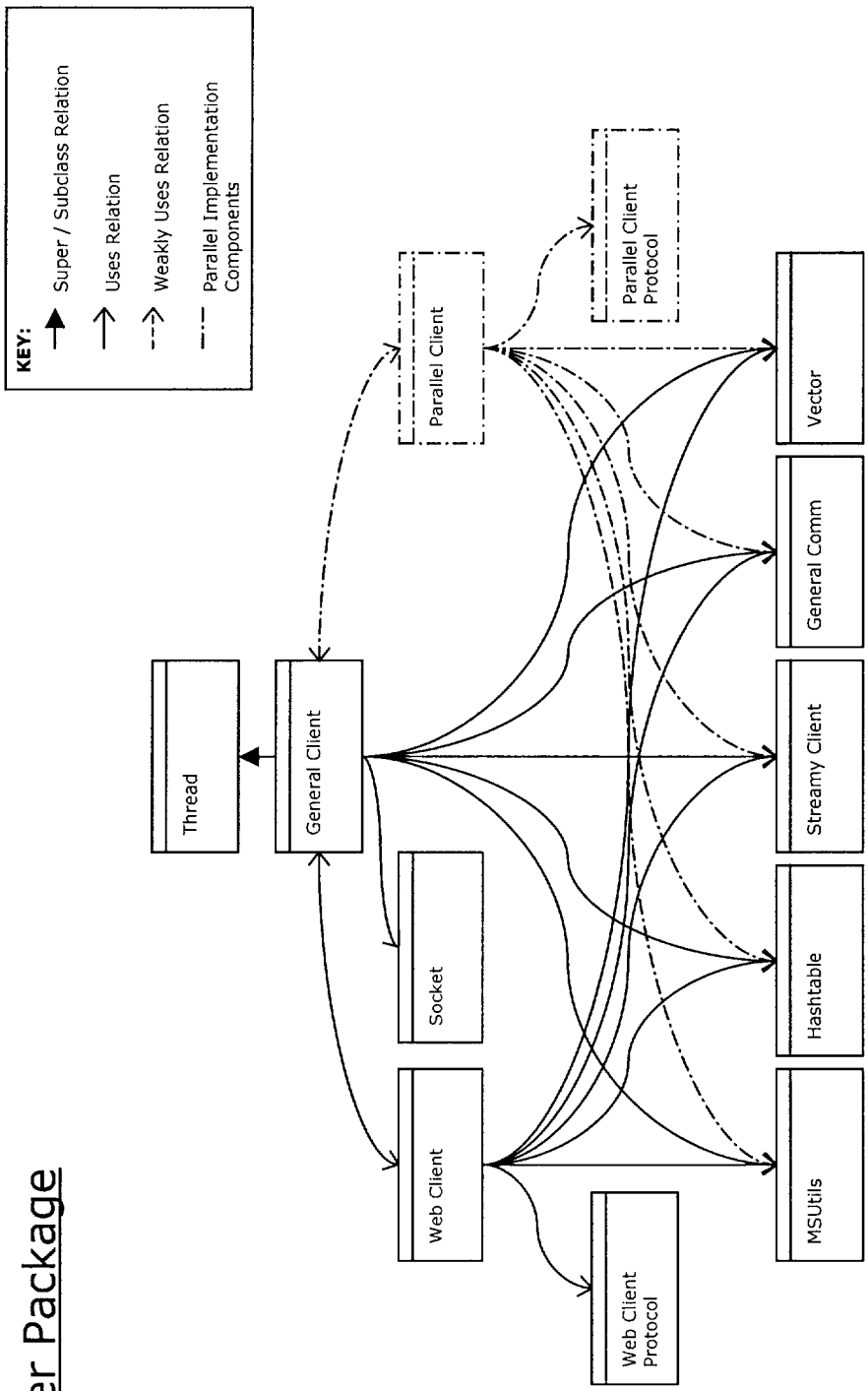
WebClient Package

Parallel Only

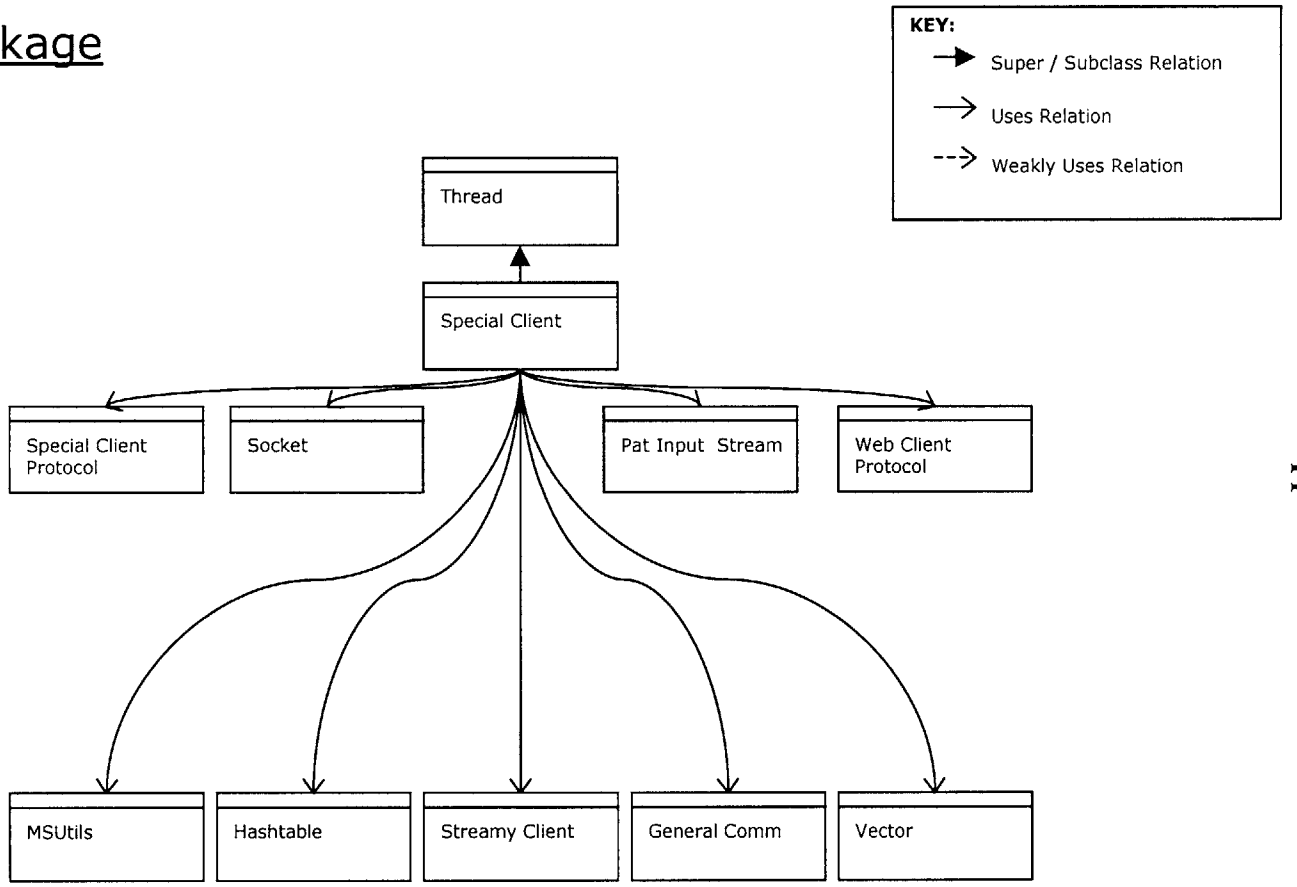


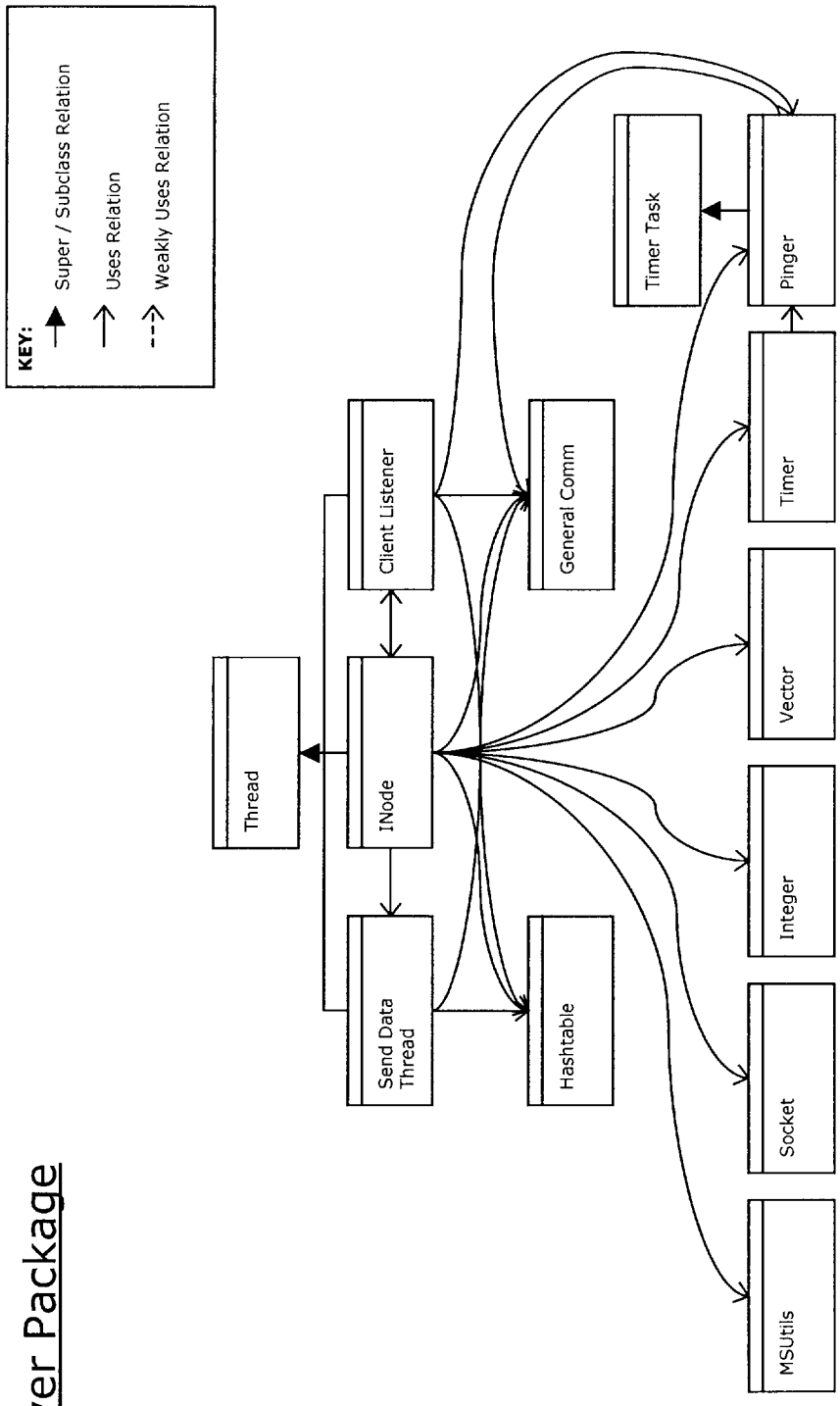

KEY:
Main Package to Support Packages Relation
Inner Classes Relation
Parallel Implementation Components
Server Package
Pinger
Client Listener
Send Data Thread
INode
Special Client
INode Cluster
General Client
Parallel Client
Control Center Server
Web Client
java.util
java.io
java.net
java.awt
javax.swing
javax.media.jai
mediashare. mail
mediashare. utils
mediashare. protocols
piw.opat. threehouse
Multi Purpose Client Thread
Multi Purpose Checker
Data Stack Thread
New Job Stack

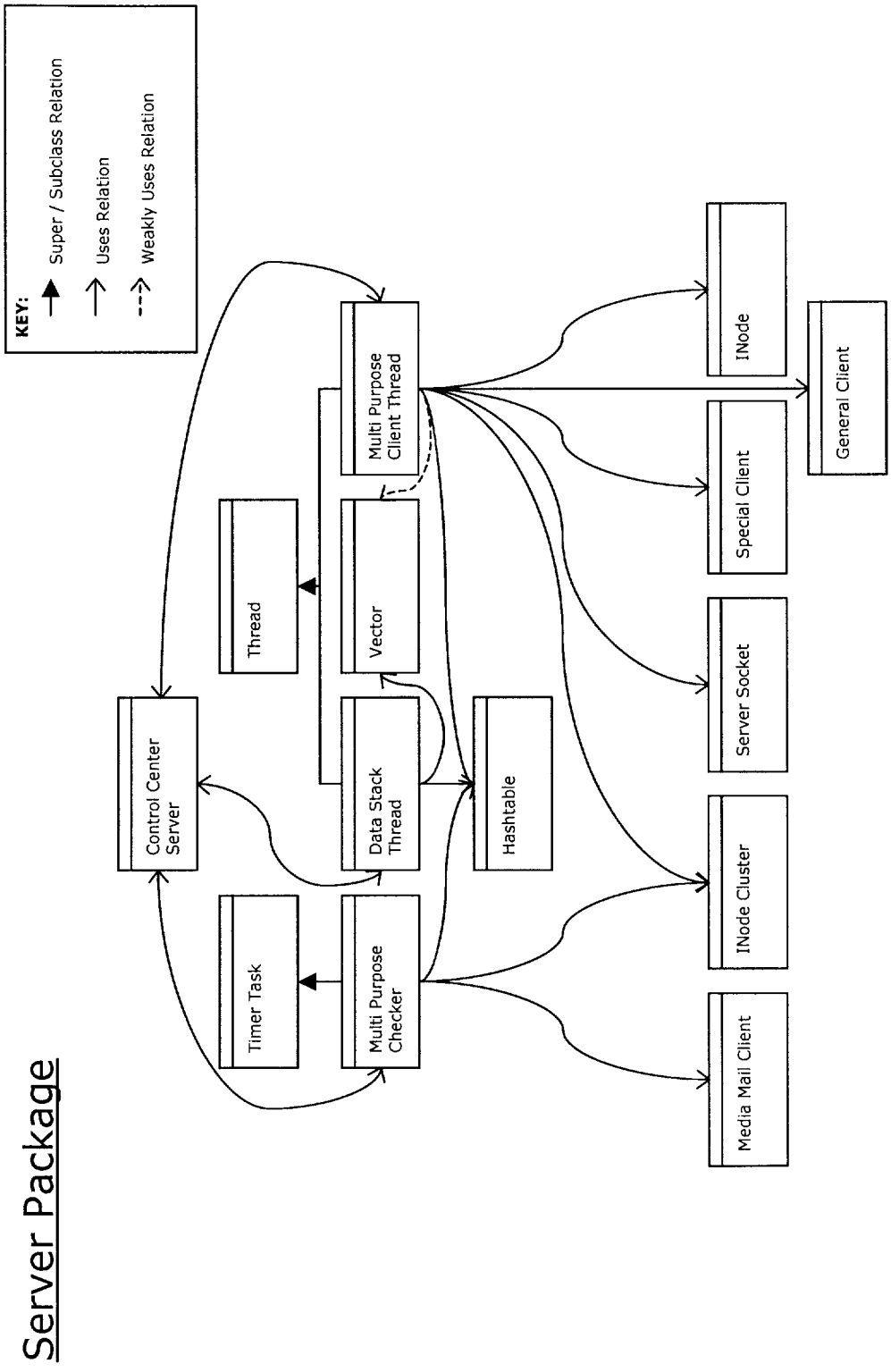
Server Package

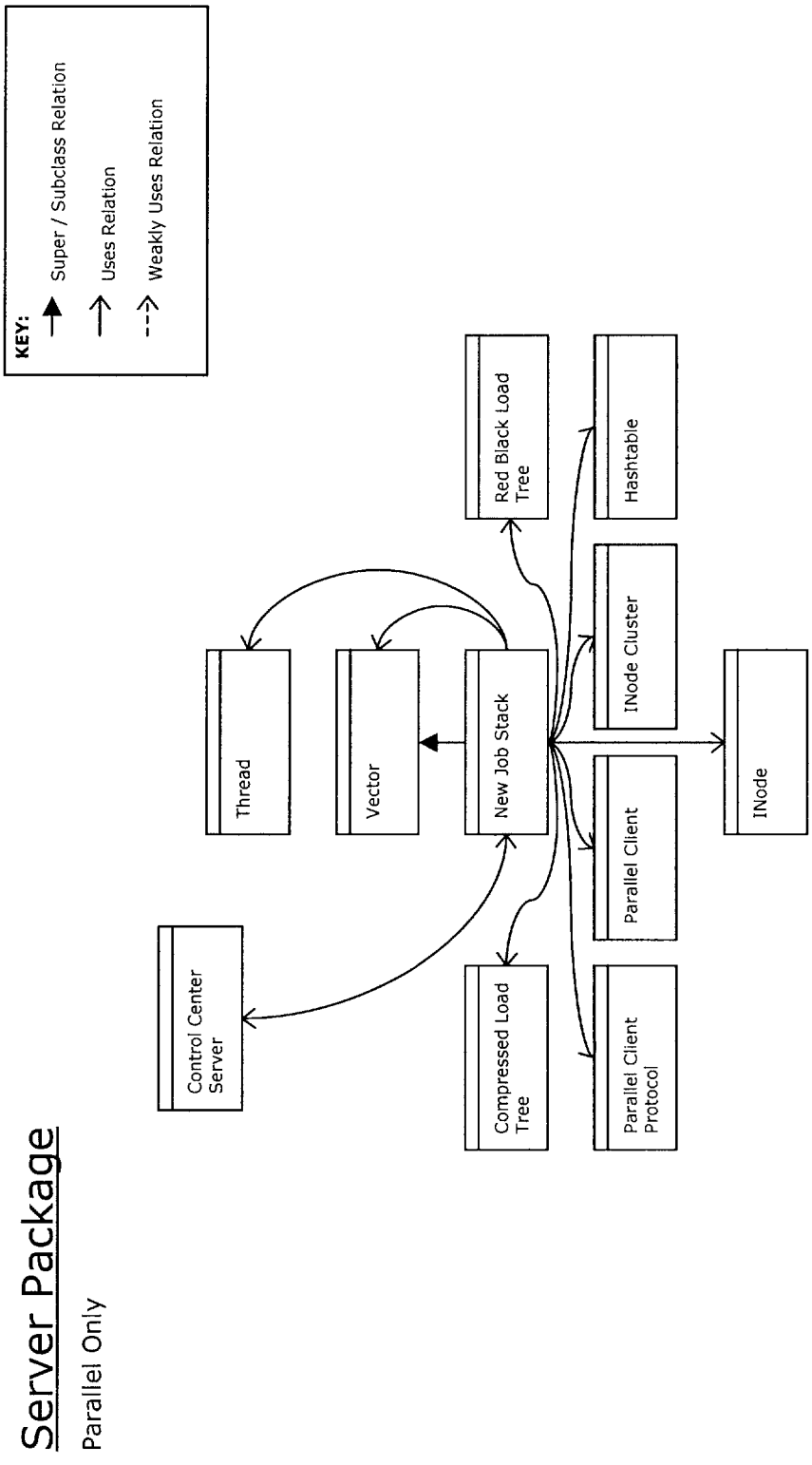

KEY:
Super / Subclass Relation
Uses Relation
Weakly Uses Relation
Parallel Implementation Components
Server Package
Thread
General Client
Parallel Client
Parallel Client Protocol
Socket
Web Client
Web Client Protocol
Vector
General Comm
Streamy Client
Hashtable
MSUtils

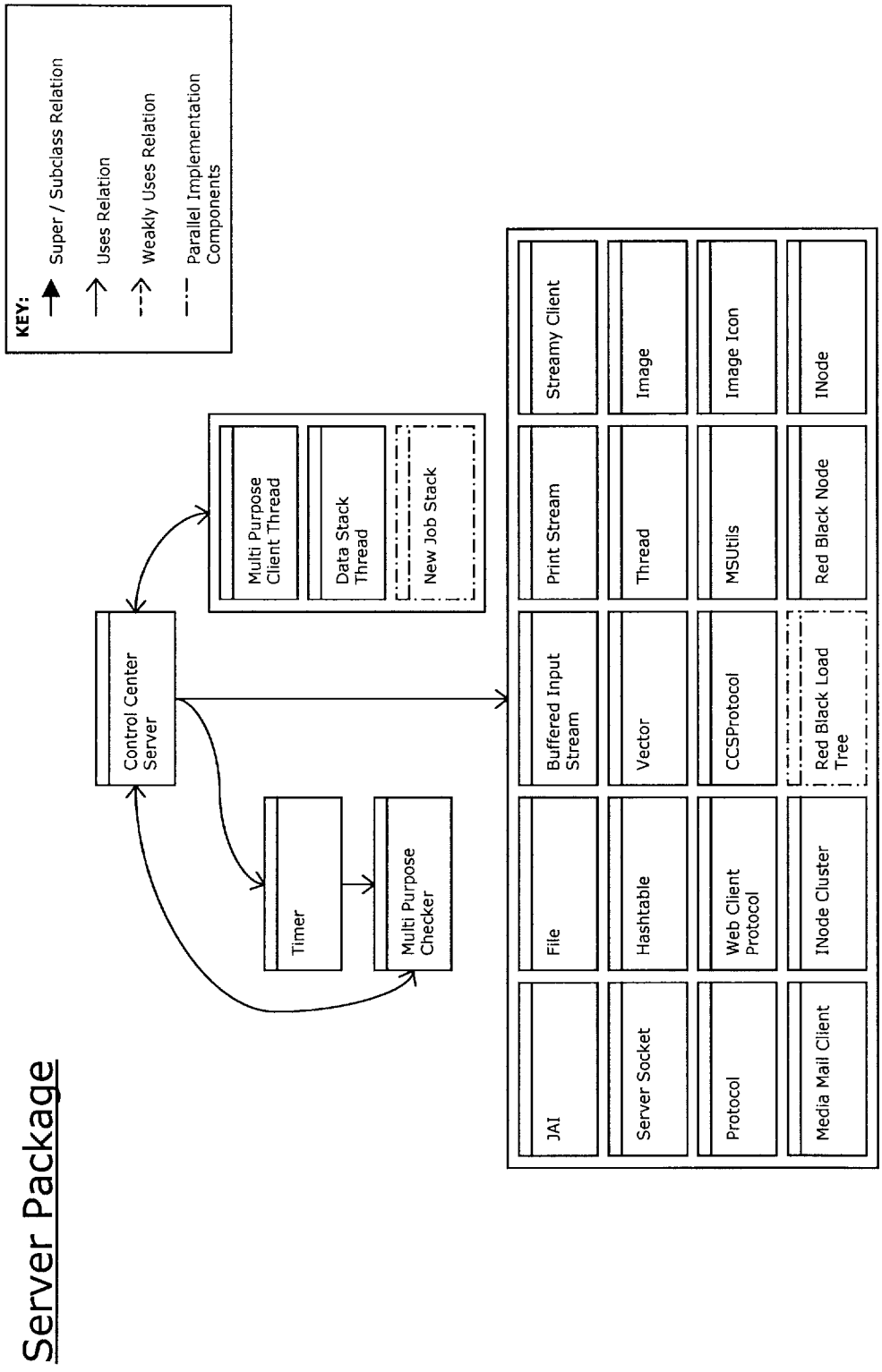
Server Package




KEY:
Super / Subclass Relation
Uses Relation
Weakly Uses Relation
Server Package
Thread
INode
Client Listener
Send Data Thread
General Comm
Hashtable
Timer Task
Pinger
Timer
Vector
Integer
Socket
MSUtils

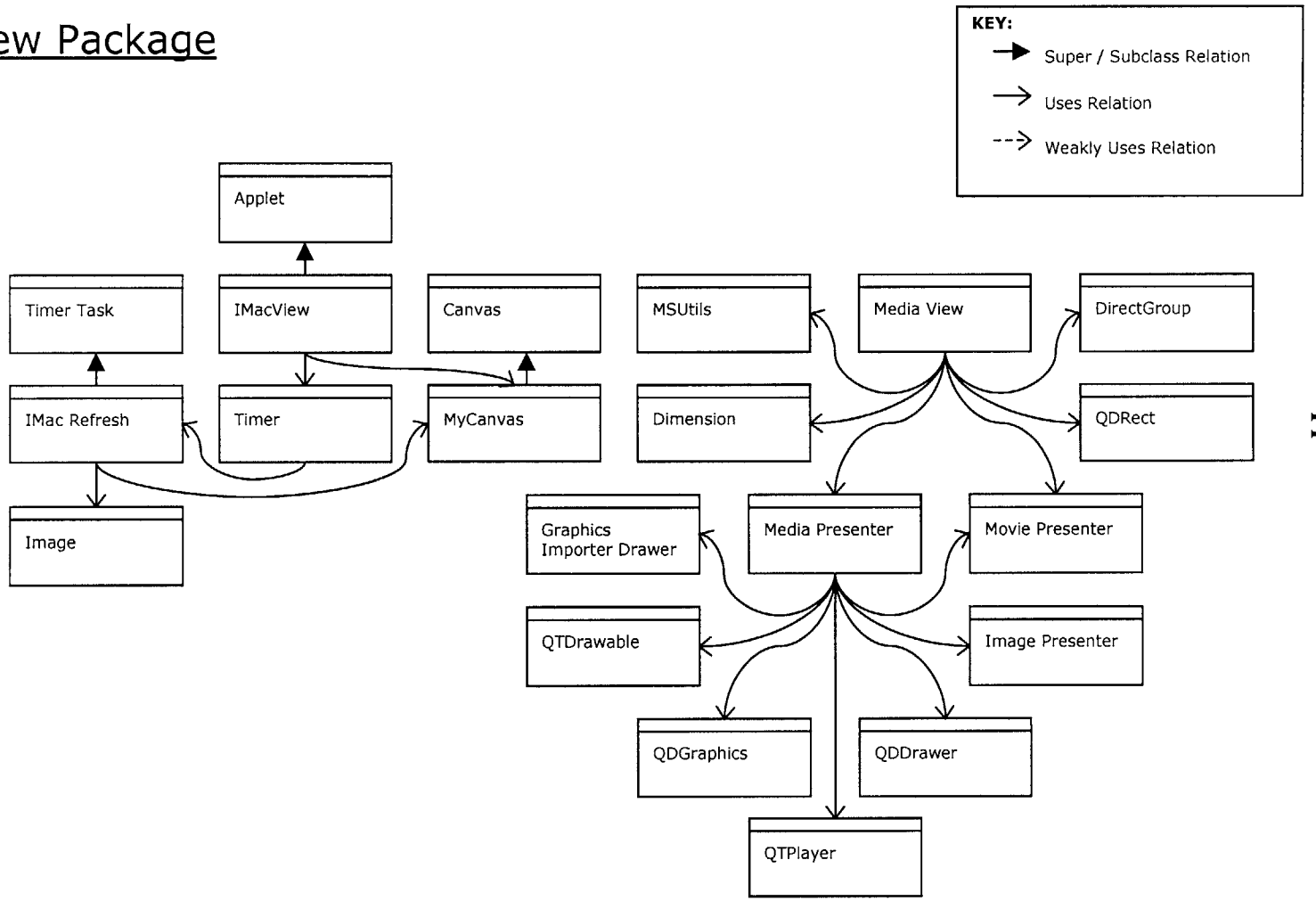

KEY:
Super / Subclass Relation
Uses Relation
Weakly Uses Relation
Server Package
Control Center Server
Thread
Vector
Multi Purpose Client Thread
INode
General Client
Special Client
Server Socket
Hashtable
Data Stack Thread
Timer Task
Multi Purpose Checker
INode Cluster
Media Mail Client

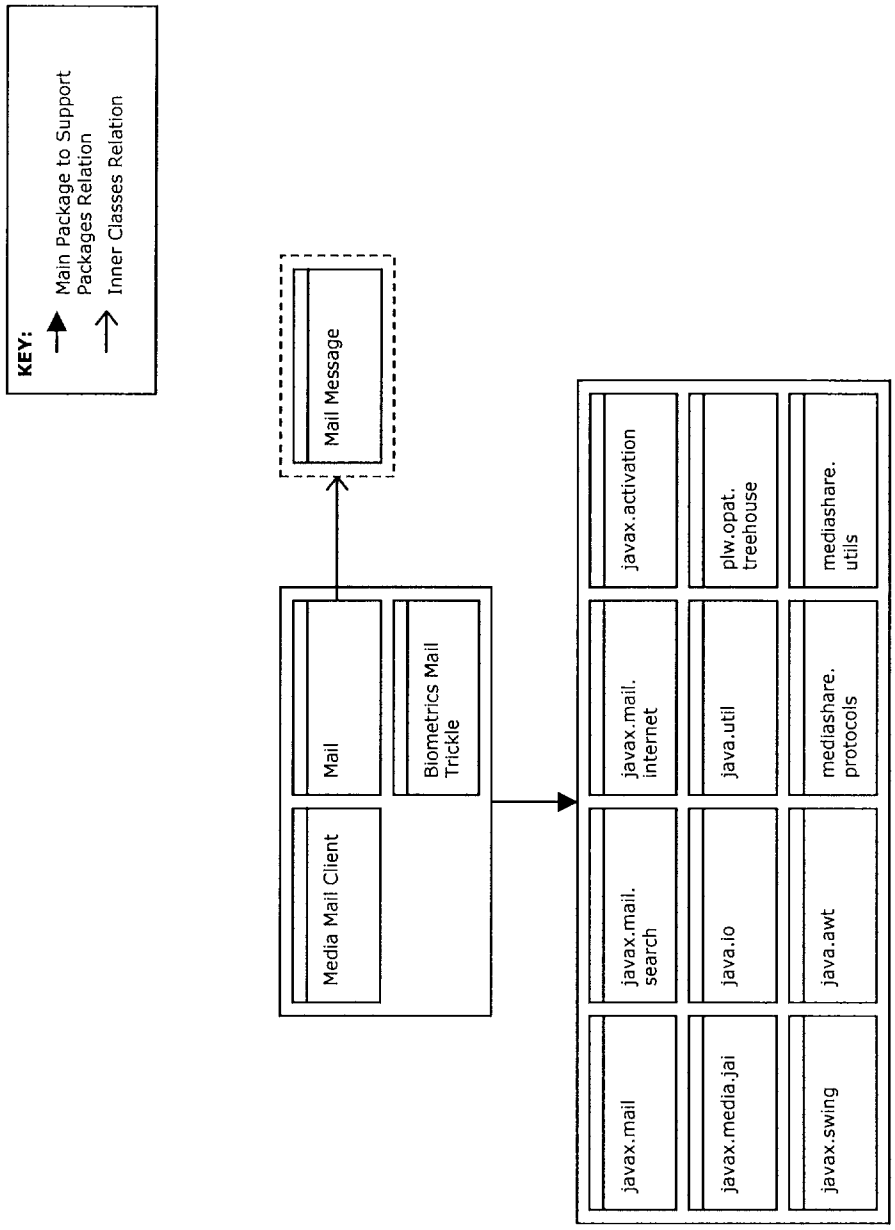

KEY:
Super / Subclass Relation
Uses Relation
Weakly Uses Relation
Server Package
Parallel Only
Control Center Server
Thread
Vector
New Job Stack
Red Black Load Tree
Hashtable
INode Cluster
INode
Parallel Client
Compressed Load Tree
Parallel Client Protocol


Server Package
KEY:
Super / Subclass Relation
Uses Relation
Weakly Uses Relation
Parallel Implementation Components
Control Center Server
Multi Purpose Client Thread
Data Stack Thread
New Job Stack
Timer
Multi Purpose Checker
JAI
File
Buffered Input Stream
Print Stream
Streamy Client
Server Socket
Hashtable
Vector
Thread
Image
Protocol
Web Client Protocol
CCSProtocol
MSUtils
Image Icon
Media Mail Client
INode Cluster
Red Black Load Tree
Red Black Node
INode

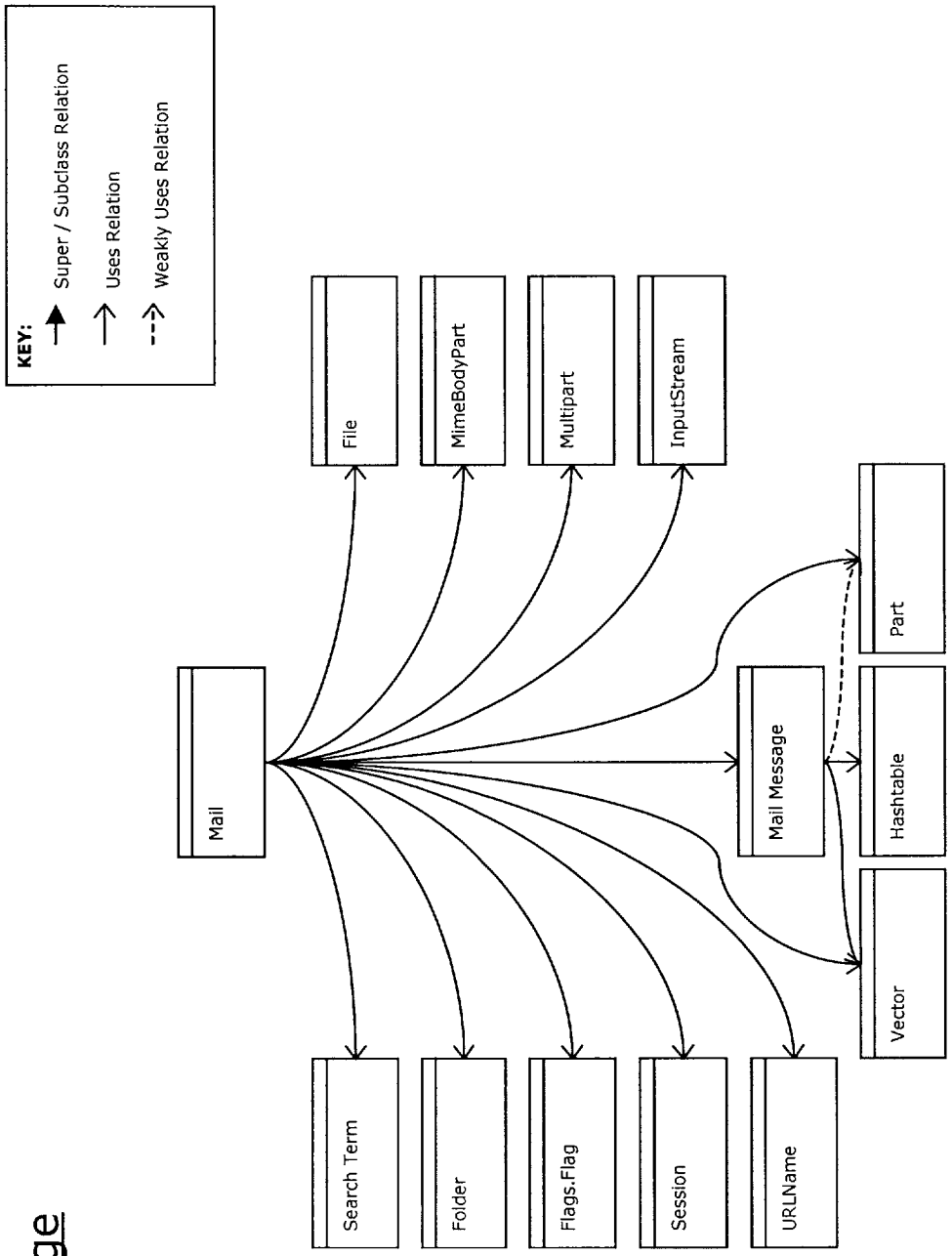
View Package

View Package

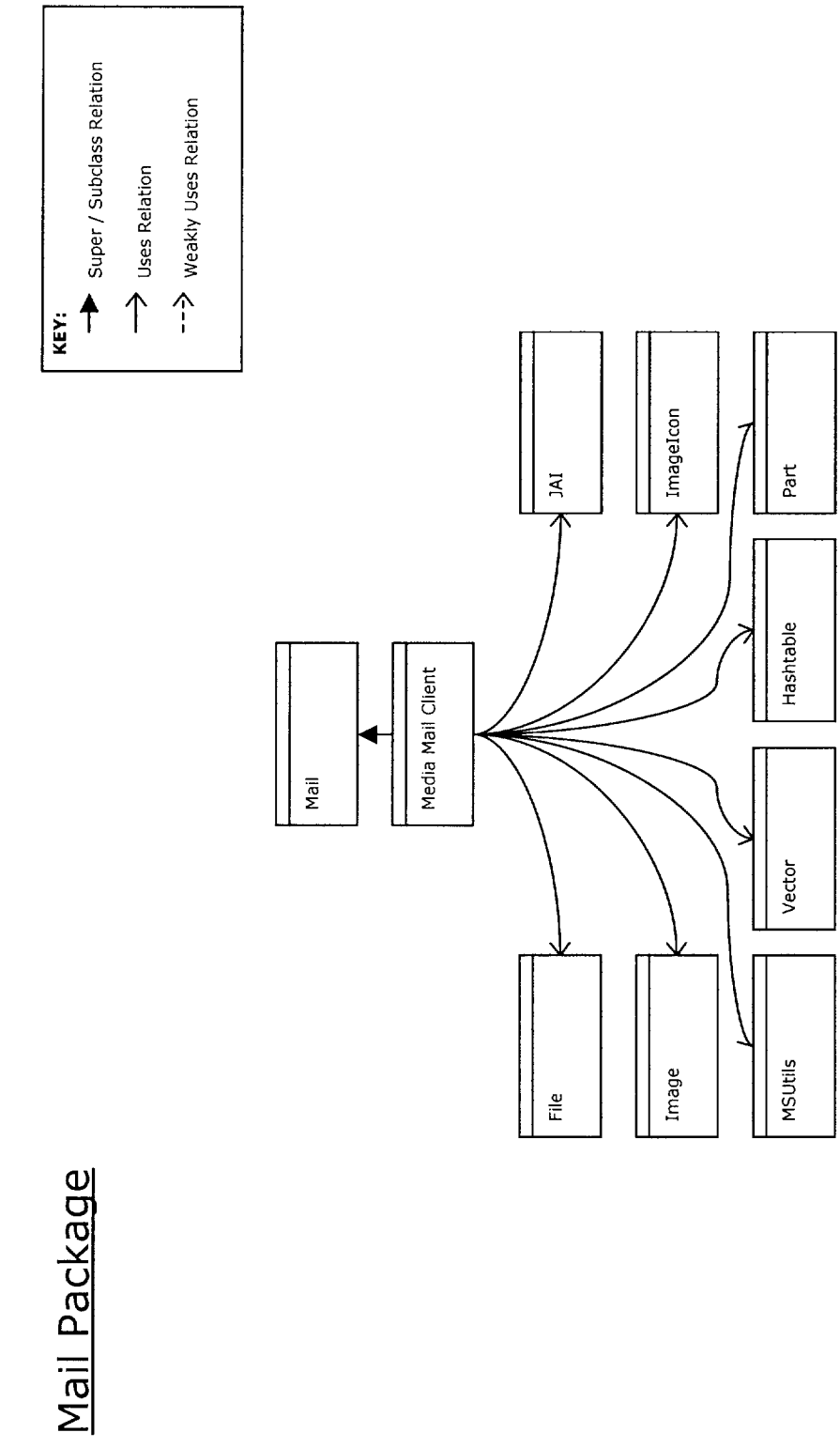


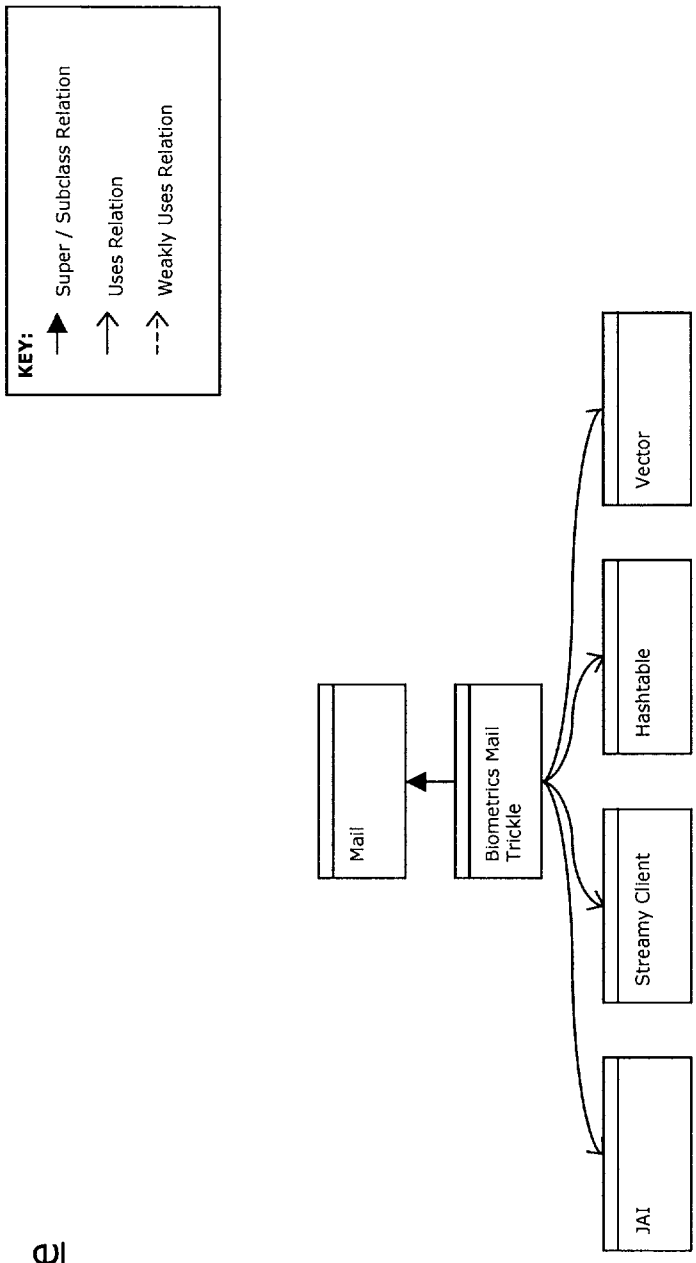


Mail Package

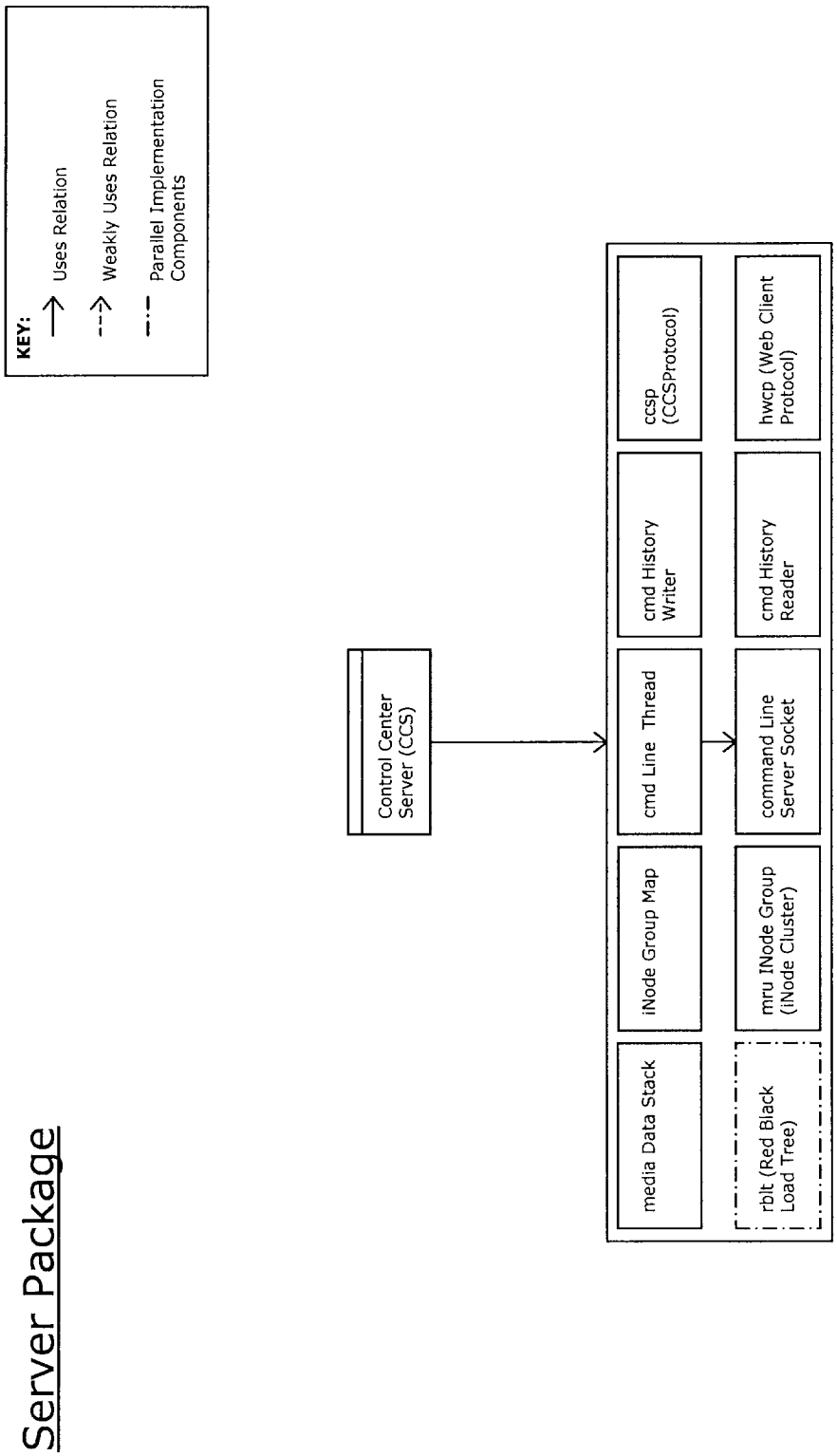


Mail Package

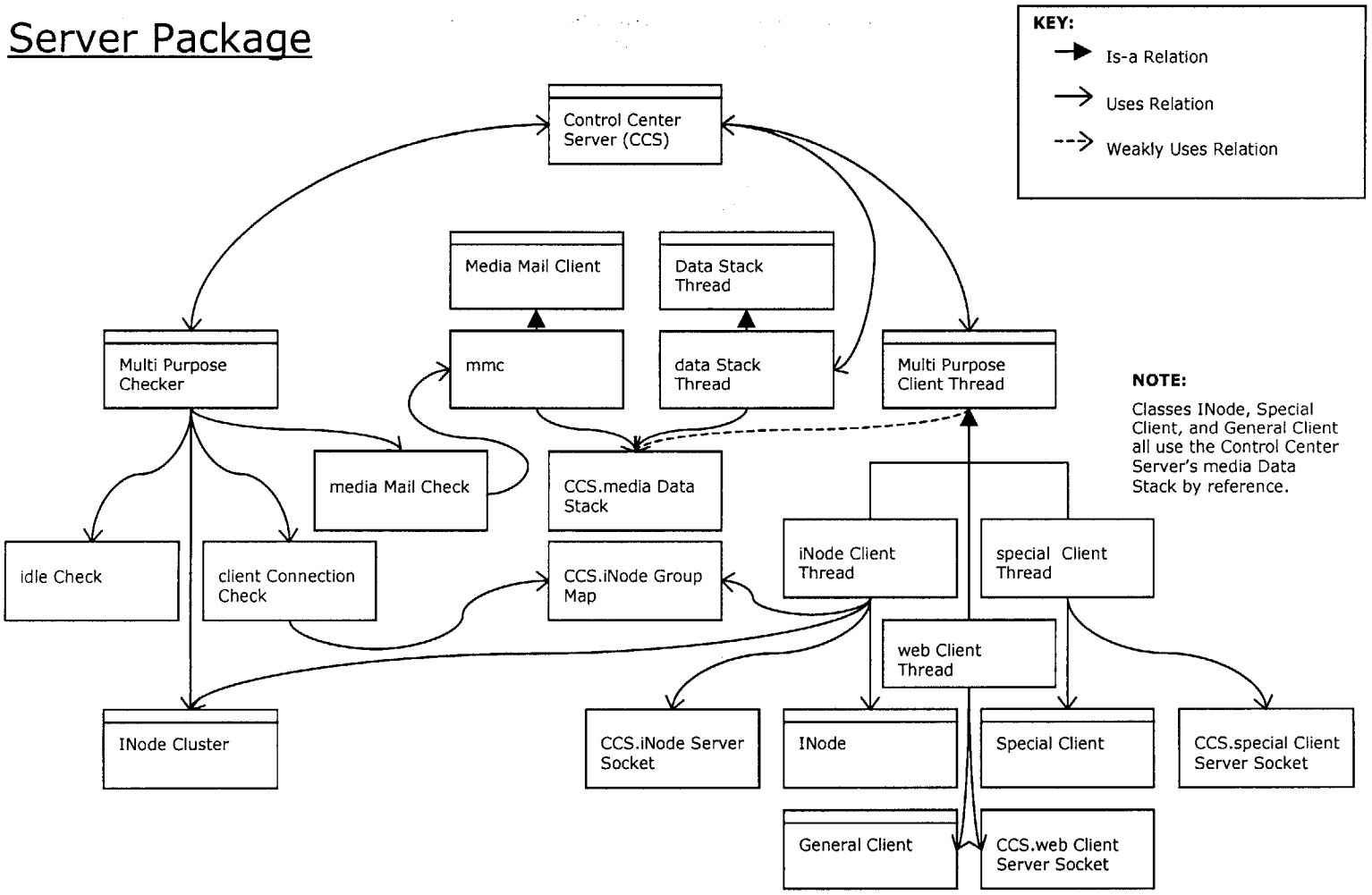

KEY:
Super / Subclass Relation
Uses Relation
Weakly Uses Relation
Mail Package
Mail
Media Mail Client
JAI
ImageIcon
Part
Hashtable
Vector
MSUtils
File
Image

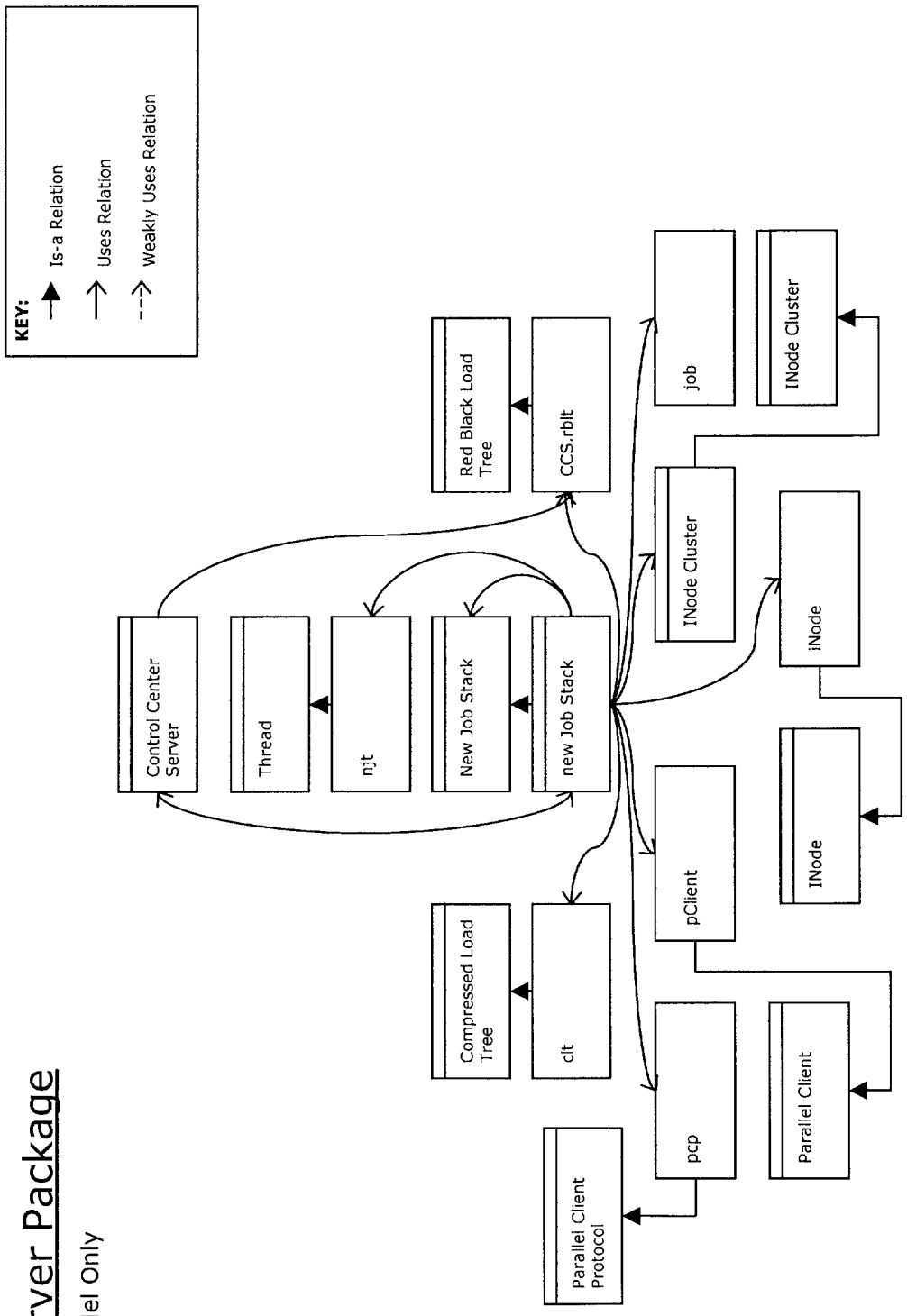


Mail Package

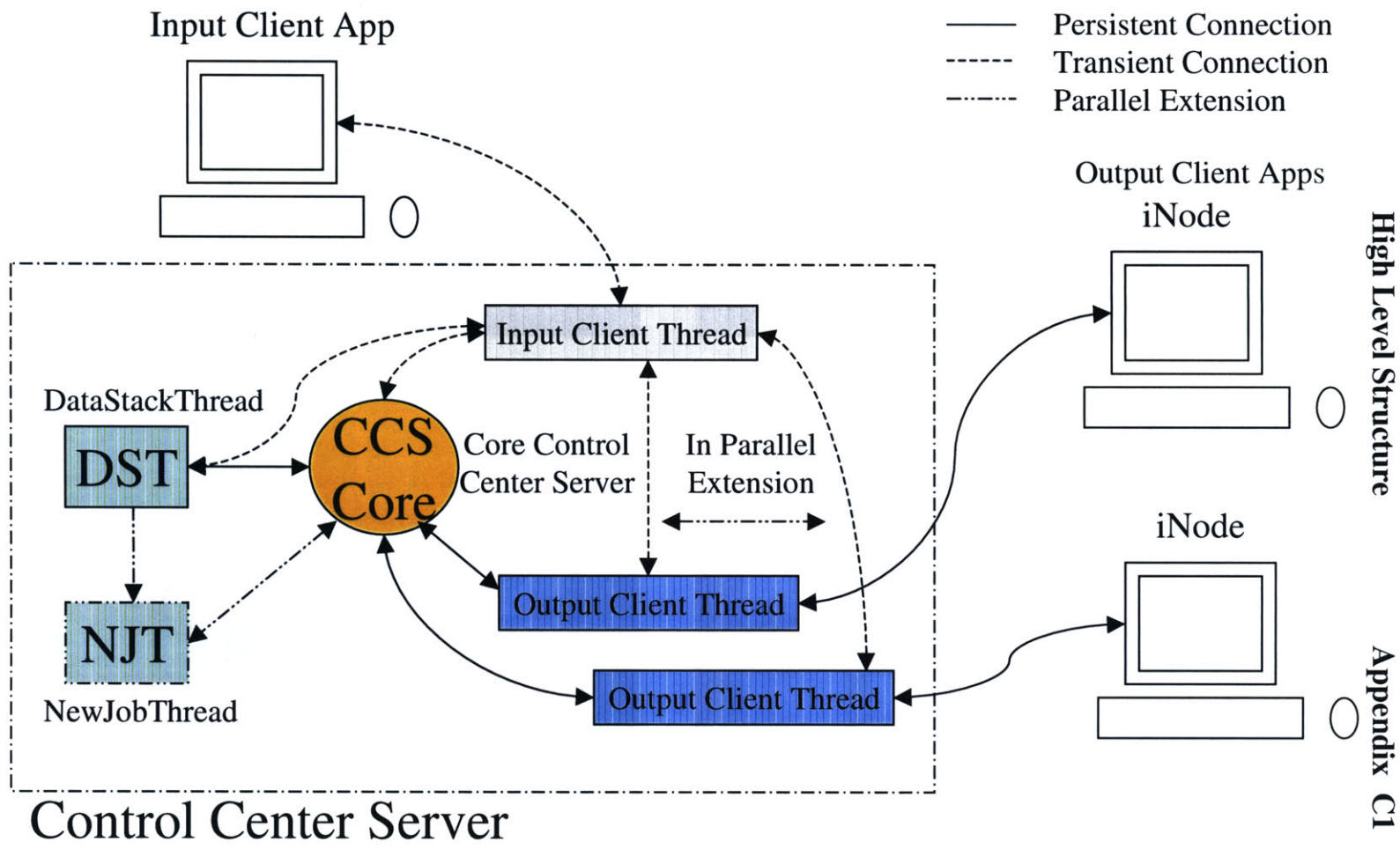

KEY:
Uses Relation
Weakly Uses Relation
Parallel Implementation Components
Server Package
Control Center Server (CCS)
cmd Line Thread
cmd History Writer
ccsp (CCSProtocol)
command Line Server Socket
cmd History Reader
hwcp (Web Client Protocol)
iNode Group Map
mru INode Group (iNode Cluster)
media Data Stack
rblt (Red Black Load Tree)

Server Package



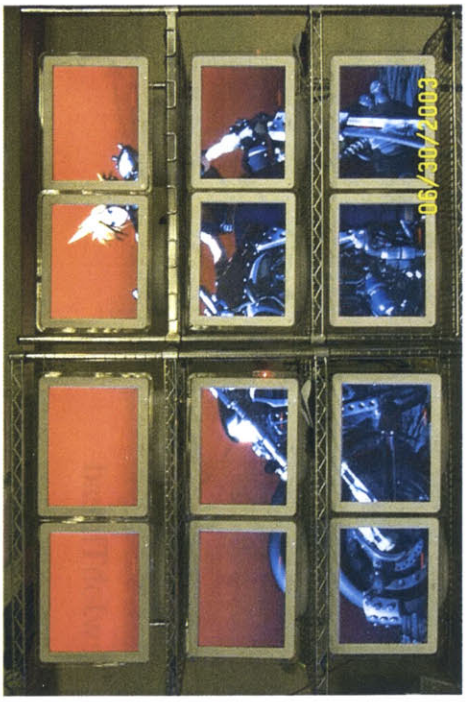

KEY:
Is-a Relation
Uses Relation
Weakly Uses Relation
Server Package
Parallel Only
Control Center Server
Thread
njt
New Job Stack
new Job Stack
Red Black Load Tree
CCS.rblt
job
INode Cluster
INode Cluster
iNode
INode
pClient
Compressed Load Tree
clt
pcp
Parallel Client
Parallel Client Protocol

Structure


Input Client App
Persistent Connection
Transient Connection
Parallel Extension
Output Client Apps
iNode
Input Client Thread
DataStackThread
DST
CCS Core
Core Control Center Server
In Parallel Extension
NJT
NewJobThread
Output Client Thread
Output Client Thread
iNode
Control Center Server


06/30/2003


06/30/2003


06/30/2003


03/15/2004