FACULTY OF INFORMATION TECHNOLOGY
Belgrade

Bachelor Thesis
Undergraduate Studies in Information Technology
Academic year 2005/2006

Thesis Title: **Relational File System – An alternative to Hierarchy Based File Systems**

Candidate Name: Dejan Štrbac, 3rd year student,
Reg. No. 30/05

15th of June, 2006

Relational File System

An Alternative to Hierarchy Based File Systems

Author: Dejan Štrbac

June 2006
Faculty of Information Technology
Belgrade, Serbia

Acknowledgments

Any similarities of this document to current, earlier and future solutions are not done on purpose. This document is a brainstorming session written down - a thinking out loud. All influencing sources are listed in section 6, “Resources”.

Foreword

This document is a Bachelor of Science thesis at the Faculty of Information Technology in Belgrade, Serbia. It is mainly about the Relational File System as alternative to hierarchical file systems, but covers actual issues in Graphical User Interfaces (GUI), Human Computer Interaction (HCI), Operating Systems (OS) and Computer Science (CS) in general.

This document assumes the reader has a fair knowledge of modern computer systems and related matter. Related research and documents used are in the chapter "**Resources**". Most of the research in this document is done from the user's point of view, and deals with underlying, today's trivial concepts, with the aim of improving usability for the next generations. Therefore, this document deals only with concepts, discussing them, and providing alternatives, so that they can be shaped according to nowadays' needs. The next step, the implementation should occur only when the concept is mature and proven.

Thanks to everybody who supported this paper and especially Ph.D. Ranko Babić for his comments and suggestions.

Уводна реч

Овај докуменат је дипломски рад на Факултету Информационих Технологија у Београду, Србија. Садржај је углавном о релационом фајл систему као алтернативи хијерархијским фајл системима, али уједно додирује актуелна питања графичких корисничких интерфејса (GUI), интеракције човек-рачунар (HCI), оперативних система (OS) и рачунарске науке уопште.

Овај докуменат подразумева да читалац поседује одређено знање модерних рачунарских система и везане материје. Везана истраживања и документи коришћени, наведени су у поглављу "**Resources**". Већи део излагања у овом документу је са корисничког аспекта, и обрађује подупируће, данас тривијалне концепте, са циљем побољшане употребе и интеракције за будуће генерације. Управо због тога, овај документ обрађује само концепте, коментаришући их и наводећи алтернативе, како би исти могли бити обликовани према потребама садашњице. Следећи корак, имплементација, се треба предузети само и тек тада када је концепт зрео и доказан.

Захваљујем свима који су подржали овај рад, а нарочито Др. Ранку Бабићу на његовим коментарима и предлозима.

Abstract

While designers and developers of today try to solve interaction problems on higher levels of abstraction, thus blaming the GUI for the same, no one dares to reevaluate the underlying concepts that bound those layers. One of those underlying concepts is the hierarchical file system found on every personal computer of today. Although fully functional, its interaction model, the way it communicates with and serves the user thru its hierarchical structure, has not changed since its first implementation. Fast data acquisition and file versatility in the Internet era of today, are the new major file system's requirements, as browsing and navigating are slowly becoming terms of computer history. They simply take too much time.

This document addresses substantial interaction issues in personal computing, thus suggesting possible solutions for them in a form of own relational file system's concept and own use methods.

This document's purpose is contribution to better file systems; more effective computing; document centric computing versus application centric; encourage real computer science inventions that we lack; moving readers imagination and urge for a new scientific technique of concept reevaluation.

Сажетак

Док дизајнери и програмери данашњице покушавају решити проблеме интеракције на високим слојевима апстракције, успут кривећи графички кориснички интерфејс, нико се не усуђује да преиспита подупируће концепте који ограничавају те слојеве. Један од тих концепата је хијерархијски фајл систем, присутан на сваком личном рачунару данашњице. Иако потпуно функционалан, његов модел интеракције, начин на који опслужује корисника и комуницира са њим, није се променио од његове прве имплементације. Брза аквизиција података и свестраност датотека у Интернета ери данашњице, су сада главни захтеви фајл система, а навигација и лутање по фајл систему полако постају термини рачунарске историје. Они једноставно трају исувише дуго.

Овај документ адресира актуелна питања интеракције у личном рачунарству, предлажући уз то и могућа решења за исте у форми свог концепта релационог фајл система и своје методе употребе.

Сврха овог документа је допринос бољим системима; ефективнијем рачунарству; документ оријентисаном рачунарству насупрот апликационо оријентисаном; охрабривање стварних рачунарских изума којих недостаје; покретање маште читалаца, стварање критичког става и подстицање стварање нове научне технике преиспитивања концепата.

Call to Action

Lets forget everything we know and get back to basic concepts. Original computing concepts have been designed by hobbyists, a long time ago, and have not been changed since. Large commercial developers will not invest money in developing and deploying new concepts, as they might not sell well as current. Therefore, well-known, proved ideas are improved (minor functionality, major looks) and sold repeatedly. It is on to hobbyists to change the computer science flow, again.

Позив на Акцију

Заборавимо све што знамо и вратимо се основним концептима. Првобитни рачунарски концепти су створени од стране хобиста, давно, и нису се од тада променили. Велики комерцијални развојни тимови неће инвестирати у развој и ширење нових концепата, јер постоји ризик да се они не продају добро као тренутни. Због тога, добро упознате, доказане идеје се усавршавају (функционалност је мање битна у корист изгледа) и продају нанова. Само је на хобистима да промене ток рачунарске науке, поново.

Contents

FOREWORD 3
ABSTRACT 4
CALL TO ACTION 5
1. INTRODUCTION 7
2. HIERARCHY BASED FILE SYSTEMS – OVERVIEW 21
3. RELATIONAL FILE SYSTEM – OVERVIEW 23
4. RFS IMPLEMENTATION SUGGESTIONS 29
5. CONCLUSIONS 36
6. RESOURCES 37
INDEX OF FIGURES 38
INDEX OF TABLES 38

Introduction
— 1 —

It has been more than 30 years since the implementation of a first personal computer file system. CP/M, one of the first disk operating systems included a primitive file system. It did not recognize directories or folders, but handled only files. However, it had different “user areas”, which were the foundation for the whole hierarchy concept. Afterwards, it was realized that individual file containers could form a hierarchy as of a tree or root structure. In this structure, a container is capable of storing files and sub containers, and those sub containers have the ability to store more of the same, thus hiding its contents from the user. This perfect hierarchy was needed in the era of small memory capacities of up to a 1 Megabyte, since the number of those containers remained small. After the initial development of the whole hierarchy file system concept, very little has changed since. Memory got cheaper, faster, enormously larger and more reliable. However, most of the limitations of the “development age” do not apply nowadays, as they changed too. Technology advanced rapidly, yet the concepts prevailed. That is not an issue only with file systems. It is a general computer science issue. Original concepts in the computer science have been slightly improved, but overall they remained the same old concepts as designed at first. We still use computers as the past two generations, and yet we program them in the same manner. Of course, incredible uses of computers have been found, but overall they are still the same. Moreover, they surprisingly look alike, after more than three decades. It is there, the monitor, the keyboard, the mouse, and the interaction models. However, the design has advanced rapidly, as computers are not hobbyists tools anymore. They are major commercial products, and visible improvements attract sales. This is shown on Figure-1 and Figure-2, which show the 1984 Macintosh 128K and the 2006 iMac (Intel) respectively.

Figure 1 – The original 1984 Mac

Figure 2 – The 2006 iMac

The major breakthrough in personal computing was the development of the Graphical User Interface (GUI) and bringing it to the end-users’ personal computers. It changed the way users think of a computer, and improved the interaction, which became visual. This way, personal computers became more interesting and easier to interact on a higher level of abstraction, using metaphors in the form of icons, buttons, and images. However, development did not follow the expected path. It did not make the users a lot more productive, yet gave them an interesting toy. In the era of textual interfaces, the user typed or punched in the command in order to accomplish his goal. There were no unpredictable virtual places or context jumps. There were no abstractions or metaphors more than those held in the file system. The principle was simple, the user "told" the computer what he wants to do. In addition, he would do just what he was explicitly told to do so. This changed with the use of the GUI's of today, where the user "shows" the computer what he needs to do. A modern GUI user holds and manipulates with so much information, irrelevant to his goal and the path to accomplish it, where the console based user focuses on the goal he needs to accomplish and the relevant set (files, places, programs and commands). If one were to replace all graphics on a typical nowadays' GUI screen with appropriate textual description, and compare it to the old textual interfaces, they would seem absurd to use, since more than a significant quantity of information is not relevant to the user's goal nor his data. The GUI way is prettier, more acceptable to the human eye and it contains less keyboard interaction, but is still not significantly more productive than the console based interfaces. It is just different. A user still has to find the file he needs, by browsing the file system hierarchy. Then he needs to open it (edit/view) by starting a corresponding application. These are the same concepts since the first operating systems, bounded by the limitations of the file system concept.

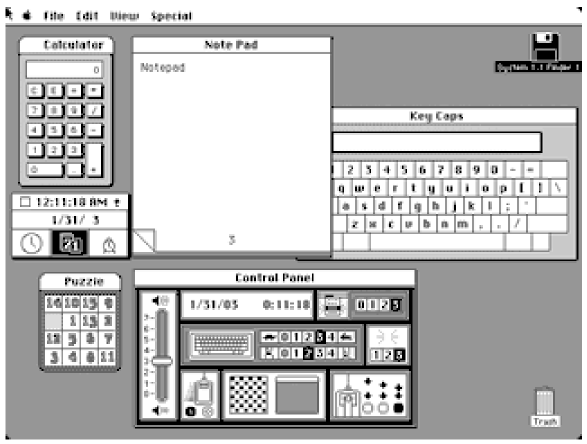

Figure 3 – MacOS desktop and Control Panel

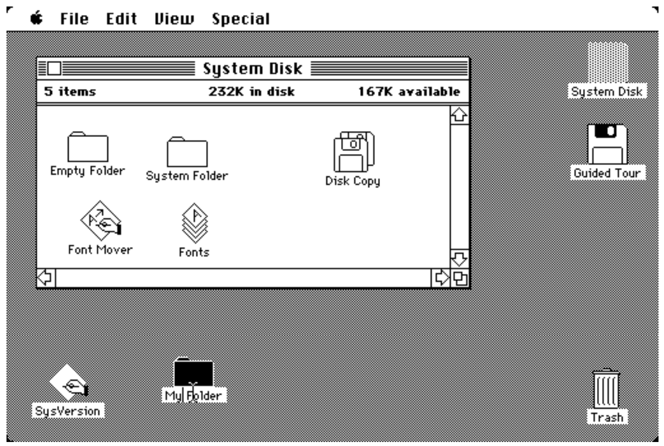

Figure 4 – MacOS file browser

Figure-3 and Figure-4 show the original MacOS operating system. Even then, in the early stages, there were too many metaphors to learn. Control Panel on Figure-3 looks rather confusing. Even the "Trash Can" in the lower right corner had to be learned at first, as its use was not intuitive. The term "Folder "became mainstream with these Macs. These Folders were represented by an icon of a physical folder, but meant a view into a file system's hierarchy. Described as such, difference between folders and windows slightly dissolves, which causes confusion. Windows OS, on the other hand, although using its representation, avoided the term Folder for a while, until recently, when everything became a folder. While almost identical, these major GUIs have not improved significantly for more than two decades. Figure-7 represents 2005 Mac OS X Tiger.

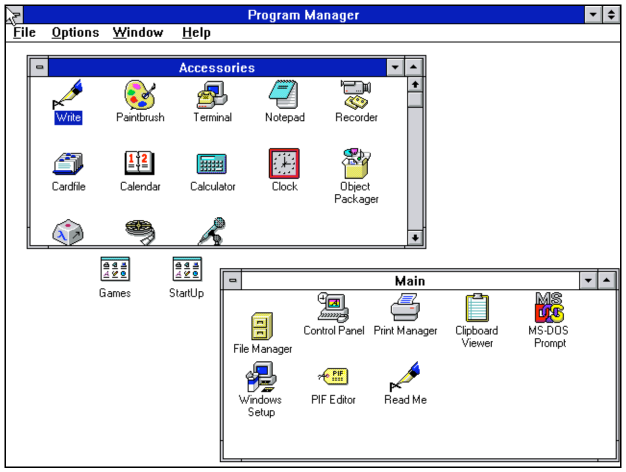

Figure 5 – A Windows 3.11 Desktop

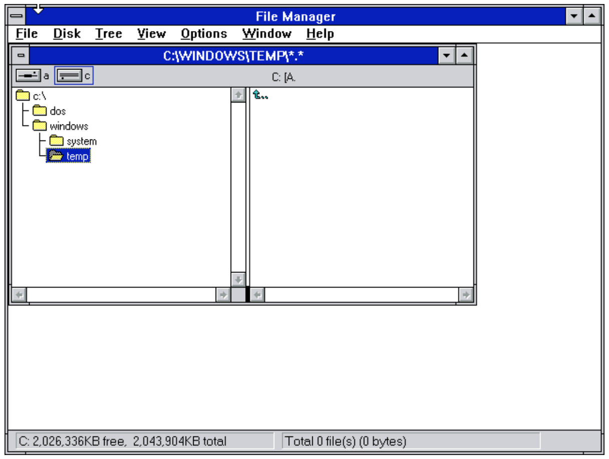
File Manager
File Disk Tree View Options Window Help
C:\WINDOWS\TEMP\*.*
C: [A
c:\
dos
windows
system
temp
C: 2,026,336KB free, 2,043,904KB total
Total 0 file(s) (0 bytes)

Figure 6 – The Windows 3.11 File Manager

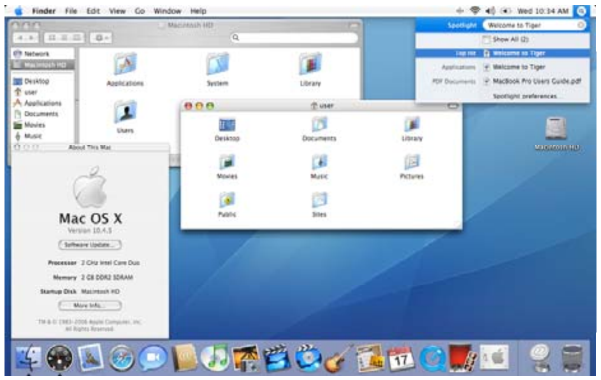

Figure 7 – A Mac OS X desktop

In today's modern, hierarchy based file systems, there can be a lot of leaves, places, subdirectories or folders as we imagine and visualize them. In the era of small memory capacities, this was truly acceptable, as the pool of places (leaves) was directly limited by the memory capacity. It was obvious that the number of leaves cannot grow to gigantic. However, there are almost no memory capacity limitations today. Many users today own memory storage devices able to handle more than 350 Gigabytes of memory. An average size of a complete electronic book is 4 Megabytes. By doing the math, it is shown that one such device of 350GB can hold about 90.000 of such e-Books, or enough for a smaller public library. How can those books be organized in a hierarchical (computer) file system? There is no acceptable way. Any organization, as such, would be inappropriate. Anything a person can speak of, has individual attributes, meaningful only to the creator (user), and is a subject to different subjective hierarchical organizations. Criterions for books organization can be multiple - title, author's name, place, subject, publisher etc. Therefore, a fixed, hard coded hierarchical organization would satisfy only one of those criterions. In such example of a digital library, the end user would have to use an indexing application in order to search for a needed book. Such application would lay on top of the file system, which would be only a gateway to the data. The file system proves unneeded and represents an extra layer that slows down the process of obtaining needed information (searched books). Its organizational services prove to be inadequate and excessive, as all the organizational logic is shifted up on to the application layer.

Taken in consideration that a hierarchical organization is physical or hard coded, it is concluded that it allows only one view on the data represented, and that is the view from the root start as represented on Figure-8. Such view has a relation count equal to the deepness of the taken leaf in the hierarchy. Each single node except the structure's root, whether folder or a file, has a fixed single relating parent. However, this is never true on any matter in the world as known. Philosophically said, everything imaginable (that we can speak of) has an unimaginable count of relations to others, as a common fact of nature. Another fact states that there are no absolute boundaries, nor limitations. As such, there is no beginning, and there is no end. Imagine the galaxy's end, or border. What is across the border? If there is a border, then something is on the other side, which cancels the statement of such galaxy border. Human brain cannot easily understand these facts, and as such, it fools self by assigning relative boundaries, relative to self or other bounded matter. Unfortunately, these human limitations have reflected on the way data is organized, representing it with metaphors, abstractions of physical things, easily understandable, touchable, visible, imaginable or just – simple.

Back from galaxies to that single node, with a single parent, lets take the example of this specific document. This document can be related to in so many different ways; single parent node is very bounding and limiting. Some examples of those relations are shown on Figure-9.

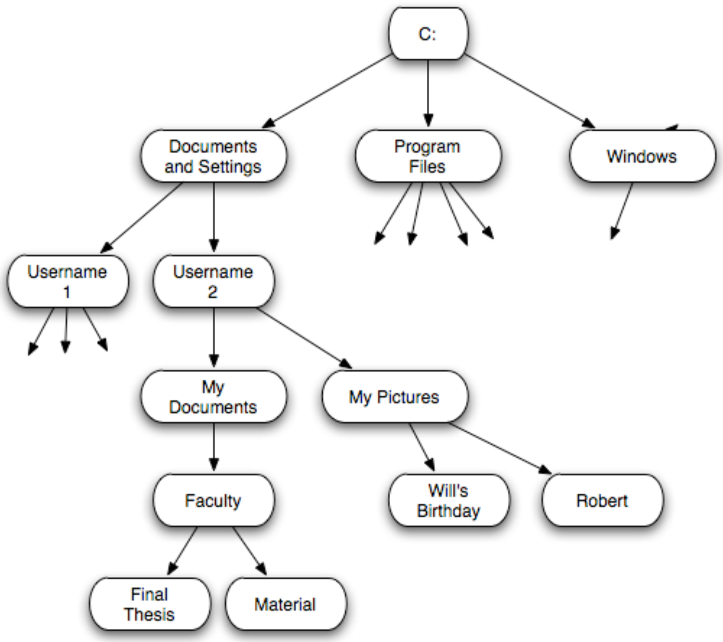

Figure 8 – Root hierarchy

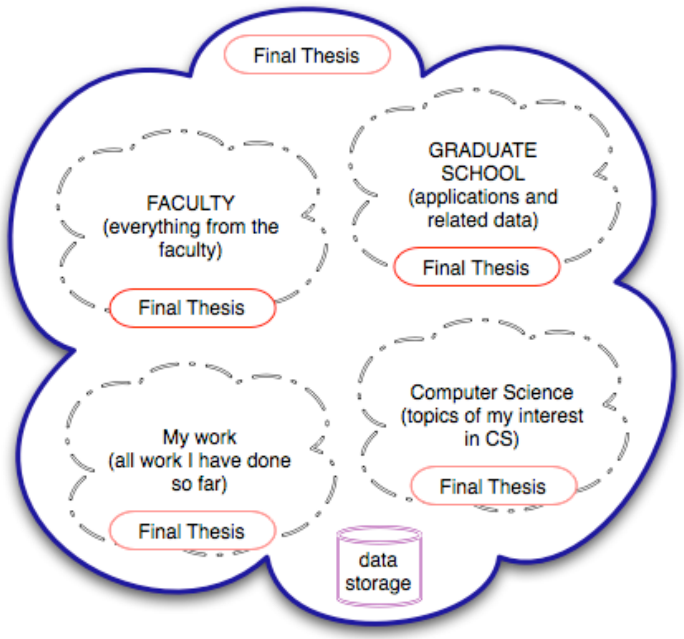
Final Thesis
FACULTY (everything from the faculty)
Final Thesis
GRADUATE SCHOOL (applications and related data)
Final Thesis
My work (all work I have done so far)
Final Thesis
Computer Science (topics of my interest in CS)
Final Thesis
data storage

Figure 9 – Multiple relations

On the other hand, the file system of today is explicitly described by using iconic metaphors. The first level of abstraction is a cabinet or a drive. Cabinet holds folders and files. Yet, those folders contain more folders and more files recursively. However, this is bad metaphor as that cannot be true in physical life. A physical folder cannot contain more than one folder, as it was not meant to. Even if it did contain one, it cannot contain two folders, as they do not fit. Moreover, the cabinet itself cannot directly hold files. It is a bad example and a bad abstraction, although somewhat easily understandable. Directories and subdirectories as used in the console-based operating systems did not have anything to do with cabinets and folders. However, that made understanding of the hierarchical structure difficult to the novice user. In the era when science is a commerce tool, this is unacceptable. Therefore, GUIs replaced both directories and subdirectories metaphors with functional visual equivalents, folders and files.

The most comprehensive knowledge of today is located in the human brain. As such, the brain has physical (relative) limitations. However, can a person himself state where his knowledge and memory begins or yet where it ends? Can a person organize his knowledge or memory in any simple static hierarchical structure? Even if a person could organize it, what would the organizational criteria be? The most applicable representation of such systems is a 3D mesh representation as on Figure-10. Data can be represented relative to other data in so many ways, and so many views. No doubt, there is functional organization in a human mind, but is it static and yet hierarchical as computer file systems are? They both store data that we need, and have methods of acquiring it on purpose.

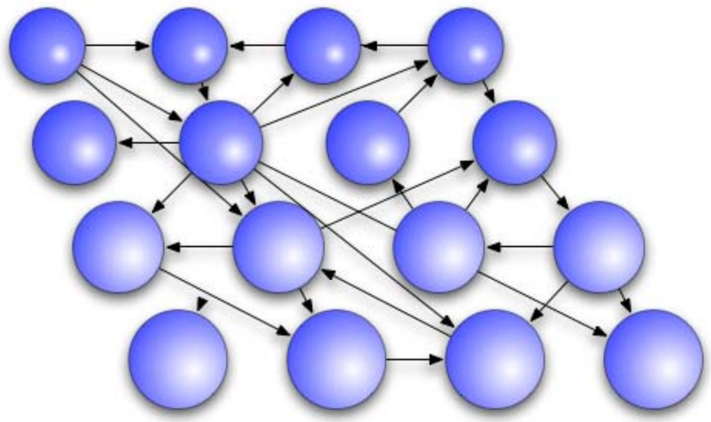
Figure 10 – A 3D mesh network

The cosmos is in chaos; chaos in which everything rapidly changes. The human world is in the same chaos. Our minds are in the same chaos. It is a natural state. However, everything relates to everything and common laws of nature prevail. This way, the world goes on, and our brains think and therefore we evolve, relating the old information to new information. A brain does not search for data and does not acquire it in the linear manner found on today's computers. Some primitive sort of search is done when some relations are broken (forgotten). If information does not relate to anything, or at least nothing important, it is easily lost and forgotten. As noticeable, the relation is as important as is the information related.

It is unnatural to make hierarchy in complex free-form information systems such as the computer file system, since that reflects on to a complex interaction with the same. Moreover, dynamic hierarchies are hard to remember, since it is not the natural way of a human. Hierarchy is a great tool to represent a current state, and to be remembered if the state is constant. However, hierarchies fail in dynamic structures or structures that change rapidly on a no constant time basis. It is natural to represent the hardware structure of a computer or a car in a hierarchy, by using the top-down model. This way, the structure can be easily remembered and reproduced since it is static. On the other hand, such approach cannot be taken when representing guests in a hotel. Not even a small one. The structure of guests changes all the time, hence is dynamical. It is also noticeable that the first structure, of a computer, represents physical system, which changes very slowly, whereas the second one, of a hotel guests, represents an information system that changes rapidly. Nobody could ever know or learn the structure. This is where the hierarchy model fails. It just is not natural to represent dynamical structures as static. They will always be hard to remember which is a prerequisite in order to navigate through it intuitively.

Computers need to be human brains' tools, an extension to human minds, and therefore those two have to be compatible. In order to accomplish natural view to the data (natural as to a human), the data has to be represented in a way it relates to other data. A human can easily understand relations. He does not even have to remember them, as they seem natural. It has even been proved that relating methods are best in language learning, hence proving that the relation of the information to other data is as important as the information itself. The only reason why a hierarchical computer structure survived for so long is the use of user created and named relations in a form of subdirectories. Subdirectories make weak one-on-one relation, which makes some sense, barely scratching the essential concept.

File system's intended use in today's computing is to allow the user to organize his data in a manner it would be understandable, easy to memorize and manage. Otherwise, lots of computer time will be spent in browsing and navigating through the file system structure in order to find the needed data. That is, if the file name describes the data best. If that is not the case, a user would have to open random files he thinks are the one he actually needs. This is obviously taking valuable time, and effectiveness of user work is degraded. One just cannot escape browsing and navigating if we speak in a hierarchy file systems' manner.

Hierarchy based file systems are easy to understand from the computer's point of view. However, this is not true from the user‘s perspective. File systems of today focus on the machine instead of the user. Therefore, the machine handles the file system with ease, while the user has to do all the hard work of remembering where is everything. Hence, it is concluded that modern file system, while reliable and robust, do not consider the interaction and human usage. They simply shift the responsibility to higher layers, like the GUI, and expect the interaction to be magically added later. This is major mistake, as lower layers are higher layers' foundation, which work in pre-given boundaries. If user interaction has not been taken in consideration and developed conceptually on the layer beneath, it is certain it cannot be implemented on a layer above it. A GUI cannot cover conceptual mistakes of a file system. They just have to be coherent, and rely on a proven, well-designed concept.

Bad solutions and concepts, as young and green, have always been accepted spread handed. What no one expected is that the users would evolve with them, taking them as assumed, trivial, and transfer knowledge of them on to the following generations, while not thinking of correctness and basic underlying design at all. Such evolution is major rival of technology and science evolution, and on this issue, a battle is fought against nature. Current users think of commands, virtual places, files, folders, applications, application's location, organization, visualization as abstraction, interaction controls and miscellaneous other irrelevant things, whereas the users of the future think only about the information.

Therefore, there is no tolerance for a work-around when it comes to user interaction, yet they became so common, they became major features of some software. If there is no time to do things right, because of a fixed deadline, interaction models are stitched and covered as a work-around, conceptually wrong and error-prone, and yet politically called features of that specific software. Such stitched, work-around and obsolete code has popularly been named "cruft" as in electronic garbage, wrong or outdated concept [3]. This has become major tradition in the software and hardware industry, as it leaves space for major corrections and patches, new versions, everything that raises revenue. Although not morally right, it is good business politics, which leaves the users with unfinished software (or hardware) product, that costs a lot of money, yet it will be obsolete in the next version that comes out right after the first one's boxed sets are sold out from the local store. This principle has a huge effect on the software industry. Software becomes something it is used for a certain amount of time until it bleaches like an old t-shire and thrown away. Globally, end-users always use bad software.

For instance, why were Microsoft® Windows ME™, Windows 98 SE™, Windows 98™, Windows 97™, and Windows 96™ better than Windows 95™, when all of them were almost identical, Internet Explorer™ and DirectX™ put aside? Instead of versions, all of them could have been patches, but that would not bring as much revenue as the new versions did. Such examples are many, and not only in Microsoft®. Moreover, although promised since Windows 95™ and Windows 96™ codename Cairo, as stated in [8], document driven system is still a fairy tale, more than a decade later with the new Windows Vista™ (pending 2006/2007).

Some of the "cruft" code done in Windows™ operating systems, are the Start Menu, Task Bar, Shortcuts, Save functionality, right button drag, intelligent expanding menus, Windows Explorer and many more.

Start Menu in Windows™ has been imagined as a view to the applications, yet it does not have anything to do with them except it points to some, so it does not reflect all stored applications. Moreover, if it did, Start Menu would cover the whole user screen, forcing the user to do the search for the needed application. This also happens today when user has many installed applications.

Taskbar covers the whole bottom edge of the screen even if it is empty. This edge is invaluable for pointing device such as a mouse. According to Fitt's law, edges have indefinite deepness, which makes them easily and quickly accessible. Yet, taskbar is rather confusing, and the user has to switch among tasks in order to find the one he needs, which is frustrating when many single application tasks are running. Windows™ does not make any use of other three screen edges nor any of four corners, which are again invaluable for user interaction.

Unlike the previous two, which dealt with the GUI, Shortcuts are file system's "cruft" code, a state of the art work-around. So many times has been witnessed shortcut manipulation instead of file manipulation, especially amongst novice users, so that their usability can be truly questioned. If there were no shortcuts in Windows™, Start Menu would not work, as it is a shortcut container by itself. Instead of having no difference between the file and the shortcut, shortcuts are new files with absolute path relation to the file in the structure. If one were to manipulate the shortcut, he would not be affecting the real file, even though graphical representation states the opposite. Aware of this problem, Microsoft® programmers added a message box that alerts the user that he is dealing with a shortcut, not a file, if he were to delete that shortcut for instance; a work-around. Other serious problem is file identification. Windows™ identifies files by its path instead of a unique file ID. This is witnessed with the famous "Recent Documents" in the, most commonly, File menu; if a file is to be renamed or moved on to another location on the same disk, all application references to that file are immediately broken. This obviously is not "a feature".

The Open Source community makes mistakes that are more serious than those made by major vendors like Microsoft® and Apple®. Instead of truly starting from scratch, with a goal of making better systems, open source programmers copy the interface, interaction model, look and feel and solutions from major commercial products. Moreover, they copy badly. They do everything to make the software look alike the commercial one. This is done so that users of commercial products can easily switch, most likely free, and has nothing to do with science. This, commercial thinking is major slow down for the progress of Computer Science. Not much is done for hobby, but for money. Money says that proven solutions sell. Concepts that have been drawn 30 years ago by hobbyists are still in the stores in the same package and same context. There is nothing new, no inventions, no new models and no better interaction in open source software. It is even worse compared to commercial software. It is major feature is that it is free. If there were something new in it, it would be commercial, not open source, as it would impact the software industry, hence generate large revenue. File systems are just some of many concepts copied from major commercial products like Unix™, Windows™, and Disk Operating System™ (DOS™). Therefore, same concepts (wrong) have prevailed, with their use being expanded.

The other "cruft" code found in all modern operating systems, or more precisely, not found in modern file systems is the Save functionality, as mentioned in [3]. This obsolete concept dates from the era of very small and very slow disks, mostly versatile floppy disks. These disks had very long access times, were of low I/O speed, so programmers broke the original concept in order to achieve speed. This most recognizable work-around of today is not only conceptually wrong, but physically too. For instance, if a person took a pencil and wrote on a sheet of paper, he would never be in doubt that his work could be lost suddenly. He draws the letters. Those letters are instantly saved on that sheet. There is no need to overwrite with black ink again or make a photocopy of it when the document is complete and work is done, in order to assure the written pencil will not bleach. This is the principle of instant saving, which is natural to humans. If it were not like that, humans would be truly illiterate.

A person takes a paper first and then starts writing, not the opposite. First designs also asked for filename and location, before the file could be edited as new. Today's applications allow for editing on non-existent files, until saved. All the data is in the "air", until put on a medium. Such concept is not data considerate, and will always be catastrophic when mixed with human nature of making mistakes and instability of nature (chaos state).

The original saving concept had in mind such instant saving, where every change that affected data on the screen would instantly be reflected on the stored data, most likely on a physical medium. What that means is that secondary memory storage always reflects the data in the primary memory storage, yet there is no difference between what is on the screen and what is written in the long-term memory. This concept has not been implemented up to date. Back then, slow hardware meant that for each change, there had to be a wait time measured in seconds. Such wait times are unacceptable in a computer world, especially in user interaction. I/O parallelism was hard to implement then, since memory was very limited. Even if one such parallelism were to be implemented, saving would not be instant, as the data in the primary memory would change so much faster than it is written, hence breaking the concept of data reflection. No buffering technique could balance such difference in speed. While hardware of today is significantly faster, with enormously shorter access and I/O functions times, the concept of instant saving has been forgotten. That makes all save functionalities in all operating systems “cruft” and obsolete, since the concept itself of post-save is obsolete.

In addition, instant saving has been recognized as an urgent need in computer programs, as users are loosing their unsaved work, and no real solution has been given. Microsoft® implemented the auto-save function in newer versions of Office™ using checkpoints and scheduled (timed) document saving. This is not a real solution, it is a local work-around, found only in large, expensive applications, making them somewhat robust, while it should be system wide feature, based on the underlying file system. Moreover, altered or added data since the last checkpoint will be lost if there was a system stress (system crash, program crash, power outage, power surge or similar). In today’s developing Information Society, information is more valuable than money, and such must not be lost. Therefore, information loss in computer programs is not acceptable.

Such information loss is also unexplainable to modern, non-technical users. If one enters data into the computer, he expects it is already saved as it shown on the screen. This was the principle that made now ancient typewriters famous and so widely used. Computer is a tool that should help the user accomplish his goal. If he knew that the data could be lost each millisecond, he would grab a pencil and a paper, or maybe a typewriter again. Such behavior is understandable. By observing an average computer user typing a document, we can track his expectations of a computer, his unsophisticated behavior towards it, and most important, his interaction problems. User hits a letter on the keyboard while looking at it, raises his eyes view and looks into the monitor to make sure he typed in the letter he wanted and that the computer accepted the keystroke. It is assumed it is saved, as it has been shown, not realizing it is still logical, not physical. Then he continues repeating the process. When the text-processing application (without auto-save feature) crashes, user is in shock. He does not understand what happened, does not care, and yet thinks of the input data only. He looses more time in search for the now-gone data. Once he realizes his data and time have been lost, he is frustrated and does not think of the computer as a tool in aid.

Conceptually correct solution for this problem is and should be a matter of the underlying file system. As stated before, limitations set by underlying layers such as the file system affect the complete system. Current file systems’ I/O allow for sequential and direct access only. This however is very limited and bounding with its basic routines (read, write, append). There is no possibility of dynamically handling a stored file on a secondary memory without major utilization of the system. This limits the way programs and end-users manipulate files. Example given, a part of a file cannot be deleted, and dynamical insertion into a file on specific location is impossible, unless the file is read in the system's primary memory as a data structure, altered and then rewritten onto the secondary memory. Such a dynamic file system and its I/O routines are necessary for accomplishing instant save feature mentioned above.

On the other hand, as mentioned, information loss is unacceptable, and as such, undo is a golden feature. However, undo as implemented today is a matter of application, not the file nor the file system. This means that once a file has been closed or application closed, undo history is lost, along with the clipboard (if turned off), and there is no way to go back to some previous state. It is obvious how this matters; if a user works on a document (e.g.), and has to stop for a while, hence closing the application, he is unable to revert the document to some previous state. In addition, if a file is transferred on to another computer, this history is again unavailable. Since memory is cheap, all information input (relevant) at any time should be saved, for possible later use. Data is always easier to delete then to be reacquired. Undo history should be saved along with the file [10].

A good file system should integrate fast and simple file search. The principle of searching should not be much different than those used when searching the Internet. Popularity of Google® Desktop™ proves such need. Current file systems search methods are based on wondering recursively through the hierarchy. This is undoubtedly slow and inefficient. Moreover, such search mechanisms base only on the filename. Some of them integrated searching through files (Apple® Spotlight found in Tiger™, and announced WinFS in Microsoft® Windows™ Vista™) using indexing or database methods. However, these methods' speeds are measured in seconds, sometimes in minutes. Memory being inexpensive, searching for needed data is as hard as finding a needle in hay. It is much easier finding information on the Internet that it is on locally. Browsing, navigating and directory switching can take a significant amount of time, especially if the structure is large.

As observed by [16], Windows Future Storage (WinFS) was announced as a part of the new Windows™ Longhorn in October 2003. WinFS should have "brought the power of a relational file system" to the users. However, in Microsoft® Style, its deployment was rescheduled along with Longhorn for several times, until recently announced that WinFS will not be a part of Longhorn, which in the meantime changed name to Vista. WinFS would be available as an update. This trend is obvious since the mid 90's. WinFS was known as Relational File System, and before that as Object File System, that should have been a part of Windows™ 96 codename Cairo™, as described in [8] and earlier in this document. Note that Cairo™ was on our desks a decade ago, and such file system has not been seen yet. WinFS seems to be a marketing fraud to draw attention to the next software product and later drop features. Moreover, "FS" in WinFS does not stand for File System, as it really is not. A file system simply is not something that can be added on later as an update to the operating system. WinFS is a Framework – an API, and it is intended to be a part of the next .NET Framework. As such, only applications that use this API will work in virtual "relational" manner, while others and the user, will work in the same manner, as the real underlying file system is well known NTFS. As mentioned before, interactivity is not something that can be simply added later. On the other hand, WinFS is not even intended to work with files. It includes a light version of the SQL Server and it does not handle files but data. This data is defined individually by applications and the user thru an XML scheme, so WinFS can bind the data among itself and underlying files kept by NTFS. Beta versions of WinFS have demonstrated how slow and how complicated it is. New announcements from

Microsoft® say that WinFS will integrate in Windows™ Search. If they really wanted a searching tool, they should have copied Mac OS X's Spotlight tool better. Despite WinFS's "features", Microsoft® still announces that the real relational model is yet to come [17], not in the next Windows™ Vista™, nor in the succeeding Windows codename "Fiji", but maybe in succeeding Windows ™ codename "Blackcomb", that already changed name in Windows™ codename "Vienna." By observing the time gap between Windows ™ versions, this could be in about 15 years. It is also speculated that this relational feature may be a server only feature. Conclusion is that, despite its virtual relational model, and notable, announced similarity where a single file can live in several directories, WinFS is not a file system, but an application. Moreover, it has not been released yet, and as such, is irrelevant to this document's subject.

One other common problem of today is lack of software for viewing some file. How many times has it been heard and witnessed that someone needed to download a complete program of 20MB or more in order to open some 10KB file? Figure-11 is just one example found on many web pages today. Such approach is a major hassle. Every file should have its standard view along with it, viewable on any computer, without any additional software. If someone were to download an Acrobat PDF file, he should be able to view its data in a web browser as XHTML if he does not have Adobe Reader installed.

Figure 11 – Additional software for viewing files

Taken all mentioned above in consideration, it is concluded that a file cannot be "just" a file – a simple data stream. A file is a system's structure. It should have a view. It should have a model. It should have its controller.

Software, no matter how good it is, is worthless without a good hardware support. Mainstream personal computers' hardware still looses its information on power loss or reset. This way any good software will be left without its backbone, and will be error prone. Hardware should be data-considerate and do everything in its power to preserve the data.

Taken all previously stated in consideration, it is concluded that current systems cannot be "stitched", as their foundations are conceptually bad. A new system has to be designed from scratch, which is absolutely user oriented, leaving the power in his hands.

To better describe user needs, as that is the focus of this document, a black box system will be observed. Major features to follow:

- Machine has an On/Off switch that can be used at any time by the user without any side or post effects; machine acts in such manner as a DVD player, VHS recorder, AM/FM/CD stereo, microwave, dish-washing machine, telephone, television and all common surrounding devices;
- Machine works as a solid, built as one system; Its internals are strongly bonded together, physically and logically; user does not know about internals and should not care about them; only complete system's effectiveness is taken in consideration;

- Machine's use is to help the user solve his problem and reach his goal; machine is user oriented and acts only as a machine, a tool in aid; user should not know the language of the machine, but the machine should know the language of the user;
- Machine is mobile and can be moved while running;
- Machine is intelligent, but does not ever decide instead of the user;
- Machine is universal and its use is not limited; software is expandable;
- User is not afraid of this machine nor the damage it can cause;
- Machine is reliable, and does not ever lose any information by an accident;
- Machine and its data are not affected by a power loss, surge or spike as they are assumed in a normal working environment;
- Machine focuses on the users goal and the data, not tools or technicalities; so does the user; applications, locations, paths, windows, menus, buttons, graphic and everything else do not matter; it only matters what the general user goal is, and sub-goals on the way to achieve the general one, all performed on some data;
- Machine searches and organizes data for the user, only as instructed;
- Machine has fast response towards the user;
- Machine is interactive; it asks the user (how, when) what he needs;
- User tells the machine what he needs by best describing the action or goal'
- Machine learns from the user, observes his habits, records and uses them in providing him with better experience;
- Machine protects self from malicious commands, instructions and code; machine is capable of self-repair;
- Machine protects the data from human thoughtless actions and human errors;
- Machine recognizes the user; each user has own personalized habits;
- Machine has indefinite memory and speed; software is future-resistant and not affected by underlying technology evolution;

Such described black box machine is document driven, and represents perfect conjunction of hardware and software. Not much is different in this list from current systems. Major differences are in the orientation and attitude, where the machine serves and adapts to the user, not the opposite. This black box system is this document's higher purpose, and the file system concept described is intended to be a basis for one such machine.

Conclusively, the file system described will:
- Be relational, not hierarchical;
- Be understandable for the human way of thinking;
- Be user oriented;
- Support infinite number of views on the data;
- Allow fast and complex relational searches on the data;

- Be robust, reliable and possibly self-correcting;
- Not be affected by external incidents;
- Act as a dynamic structure and allow dynamic operations on it;
- Be small, simple and fast;
- Act as a black box by itself;
- Be backwards compatible with current applications;
- Be named Relational File System or RFS as an acronym.

Hierarchy Based File Systems – Overview
— 2 —

Structure

Foundations of hierarchical file systems are directories (folders) and files. A directory is a files and sub-directories container. Although this might not be true in the physical implementation of the taken file system, directories (folders) are always files and sub-directories containers as observed by the user. A file is an object that contains data. Both directories and files are given names that are meaningful to the user. Given those names, data contained in the files is organized and categorized. This way, it is easier to keep track of the data by remembering its location (hierarchy path) and its filename. Such organization forms a two-dimensional tree structure as shown earlier on Figure-8.

URLs and paths

One of the most important features in any file system is the Unified Resource Locator (URL) subsystem. URLs allow for unique file identification by the user and applications. URL is nothing but path in the hierarchy, formed of parent directories names and the filename.

Content hiding

Directories hide their enclosed contents. This way, system files are arranged and file space is kept tidy. However, this has downsides when in a need for a specific file, since directories can contain more directories in an endless array. One has to wander thru directories in order to reveal their contents in order to find the specific file. If directories or/and the files have not been given appropriate names, such wandering might take a long time or yet even result in lost files. Local search engines might be of some help, but not even them can do miracles since their search techniques are as same as the one performed by the user – filename comparison.

Hierarchy

Each directory represents one location. Many directories are nothing but many locations. However, in a hierarchical file system, a file can be stored only at one single specific location. This unfortunately means that in too many directories, one can easily lose track of a specific file. In addition, when storing a file, one has to find the right location for it, so it can be found later by self. Traversing thru the file system structure becomes almost unavoidable in any file system operation. Other shortcoming of single file location is that a file can have a single description and meaning given by the path, parent directory and filename. In addition, most of the time, having any hierarchy is unneeded, since the last directory in the hierarchy path (last parent) describes the file best. This makes all the preceding parenting hierarchy and the extra locations excessive.

In conclusion, hierarchical file system does not consider multiple relevancies a file may have, as it allows only for one location of a specific file, as in physical world. Moreover, that single location (along with others) has to be remembered. All the hard work is to be done by the user.

Relational File System – Overview
— 3 —

Structure

Foundations of relational file systems are categories and files. Categories are nothing but sets that gather files by common attributes. They do not contain them though, not logically nor physically. They represent the relations of the files pointed to, in similar structure of a set. Therefore, categories are relations, and the files listed by the category are the files belonging to that relation. This is substantial difference from the hierarchical file systems, since they bound the files within directories, as with physical walls around them. In relational file systems, there are no subcategories, but complementary relations – complementary categories. Categories also relate to each other, describing the relationship of files within those relations. There is no limitation on the number of mutual relations, as in hierarchical file systems. These categories may be visualized as in hierarchical file systems, listed along with files, as subcategories, but they are very different. They are "living" categories, dynamic versus the static ones (directories) in hierarchical file systems. Depending of the incoming relation, its contents will vary. This is the beauty of the relation file system. It supports views on the data.

As mentioned, categories in relational file systems can list other categories along with files. These categories can be incoming and outgoing, or more precisely, categories relating-to and relating-from the current category. They can be listed together, and have visual marks to be distinguished or just be semantically separated. Listing relating to and from categories together is the more logical choice. When a category lists other categories, those enlisted categories act as filters on the files within the current category. The filter is nothing but the logical AND operation between the relations (categories) and their order does not matter. However, this is a file system browser's (navigator or explorer) preference.

To better illustrate the category/relation principle, an example category will be taken: the *Final Thesis* category. The user starts his view into the file system from this category by typing its name in the search box or by selecting it from the *WORLD*, starting category that serves as a starting point for navigating. In this manner, relational file system is backwards compatible, and those who enjoyed browsing the hierarchical file system will feel at home. Moreover, *WORLD* serves as root (/) in hierarchical file systems, and is the first category in paths or URLs.

WORLD lists only those categories that are being related-to by *WORLD*. *WORLD* does not list all possible categories and files in the file system as maybe assumed. However, *WORLD* is the default relating-from category, if none other given upon creation.

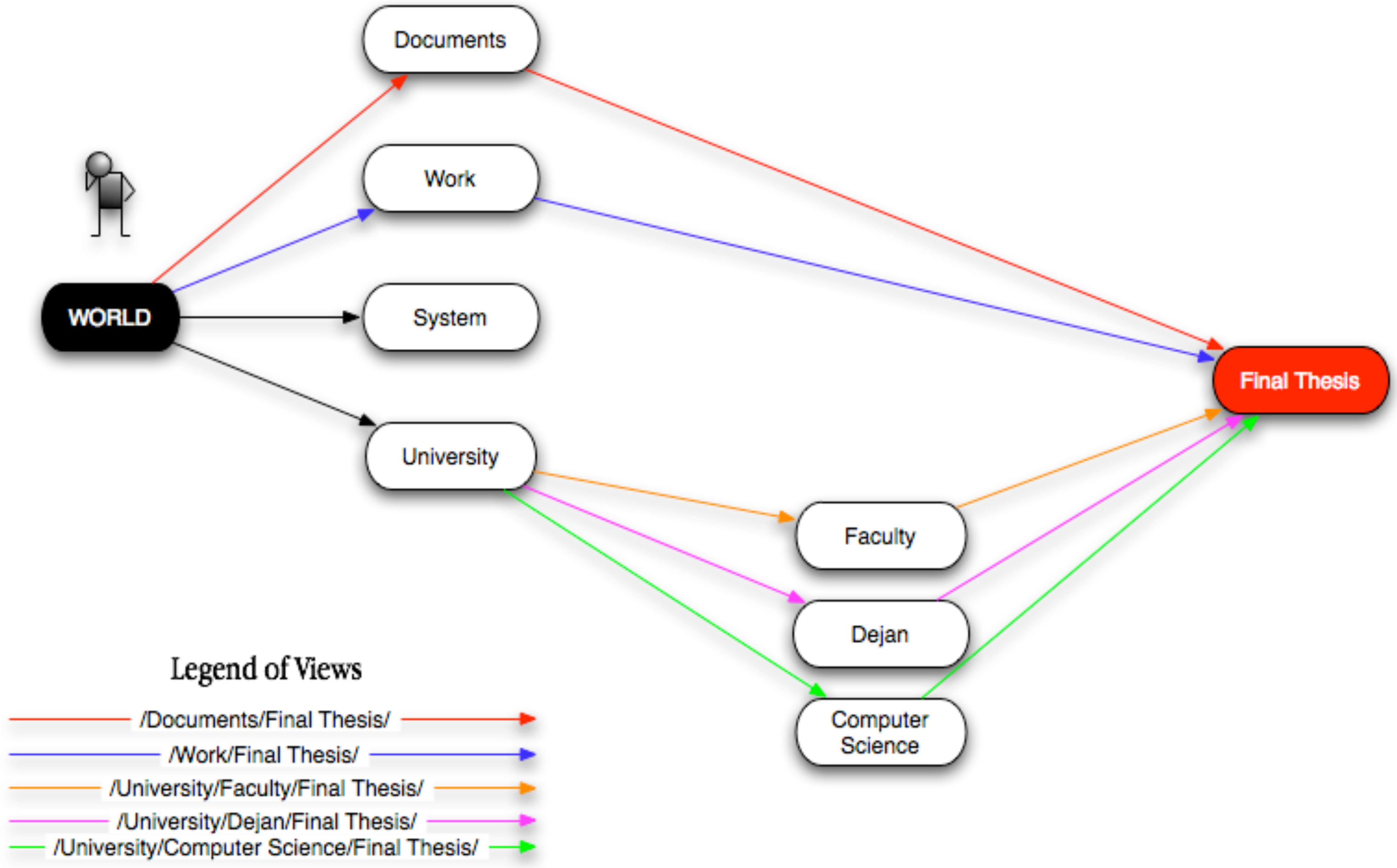

Figure 12 – An example of simple relational structure

Figure-12 represents a simple relational structure with the given *Final Thesis* category. Assumed *Final Thesis* contains files, different paths will most likely list different files. However, all files, in all listed paths will be *Final Thesis* related. For example, the red view will most likely list all Final Thesis Documents on the system, if any. This means, user's and others' final thesis files. On contrary, the pink view will most likely list Dejan's final thesis files only. The orange one will list faculty's and university's final thesis files, most likely duties, assignments and other administrational files. The green view will probably list all Computer Science related thesis files at the given faculty of the given University. However, in this case, it is assumed, the user starts browsing from the *WORLD* category – our starting point for navigating throughout the system. The situation is radically different when the user starts browsing from the *Final Thesis* category. Figure-13 illustrates this situation.

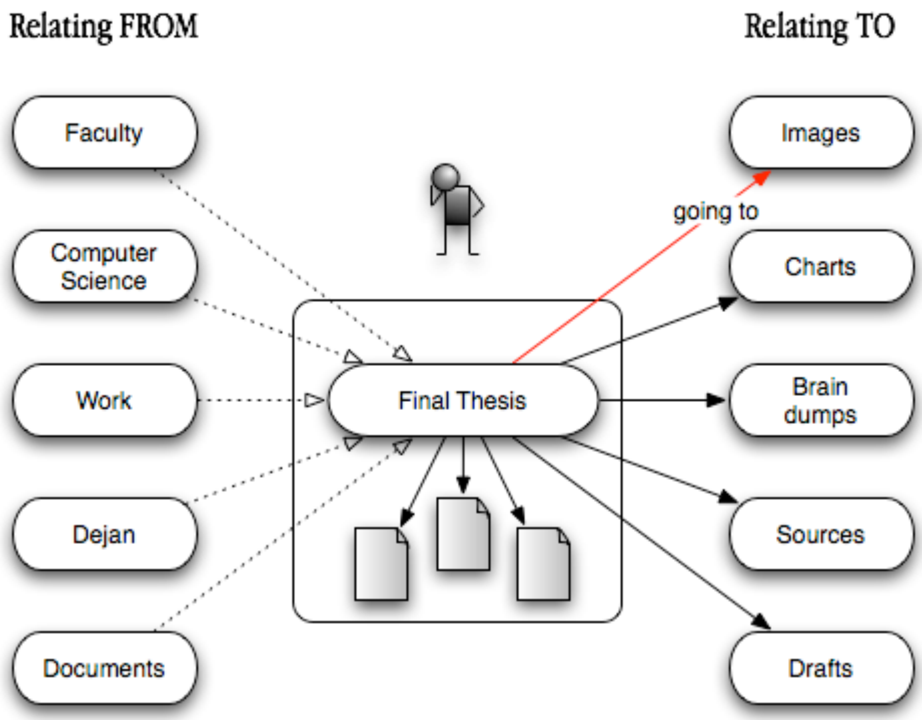

Figure 13 – State when starting browsing

Since no filters have been applied (that of relating categories), category *Final Thesis* will list all relating-to files and categories. This now leaves two possibilities. Relating-from categories can be listed along with relating-to categories, since all these relations, although not represented on Figure-12 and Figure-13 are two ways. However, this makes sense ONLY on the initial listing, when user starts browsing, otherwise will result in recursion. It is also clear that if the user goes into *Dejan* from the *Final Thesis* category, the category *Dejan* will not list the category Final Thesis again, as that would not make sense. This is backwards browsing, since the user traverses into the relating-from categories. If the user wants to go back, historically, he should always be able to use the back button (⬅) on the file system browser as in web browsers.

When forward browsing, the feeling is almost the same as in hierarchical file systems. If the user chooses the red line on Figure-13, and goes into *Images* category, a state similar of that one on Figure-14 will occur.

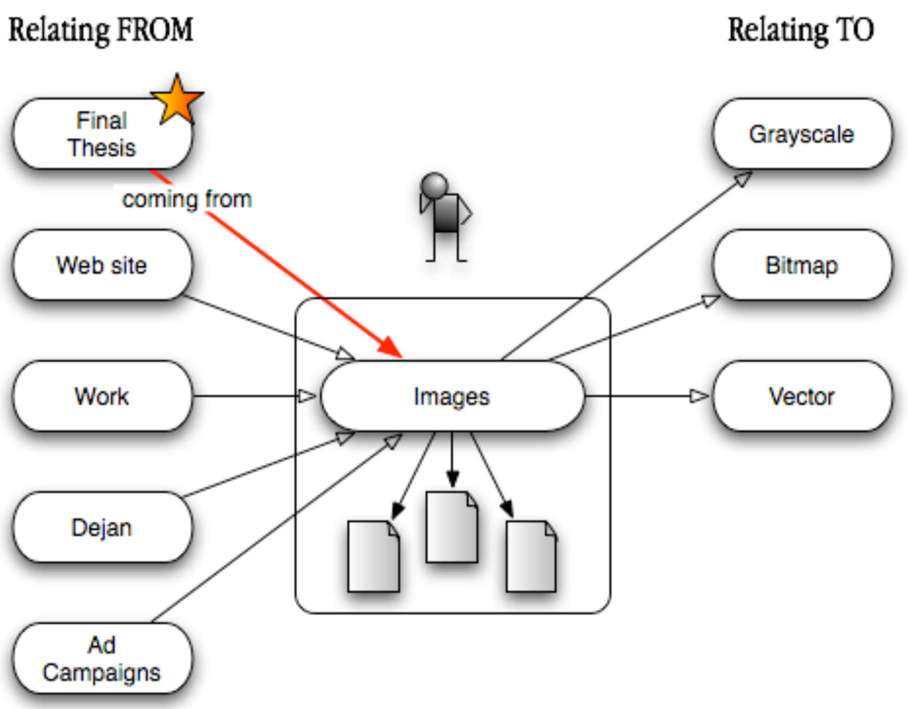

Figure 14 – Forward browsing state

Again, handling the relating-from and relating-to categories is completely at browser's freedom.

URLs and paths

Many consider that URLs are not possible in a relational model and that is downside of relational file systems, as observed by [2]. However, this document proves that compatibility is possible URLs or views are the core of the RFS hereby suggested. Because of mutual relationships, URLs are not unique anymore. There may be many different URLs pointing to the same resource. It is up to the user to chose which one he prefers. However, the resource pointed to cannot point to other than the one requested, as the file system does not allow for two identically named files in any single category. It also does not allow for two identically named categories on the entire system, as the categories, amongst other, have a filtering purpose, that of an attribute. This is not a restriction at all since categories act differently when in different paths. Again, it should be noted that categories are not containers but relations. If related in a given manner, a file may "be found" in that relation. There is no need to have two relations of same nature on the system.

Consequently, the user can forget long URLs found in hierarchical file systems. Only the last category in the URL really matters. All previous ones may be forgotten, or maybe kept only for compatibility purposes solely. When dealing with such long URLs, the previous categories will be automatically dropped. This is of great importance to the user, since finding his files just got a bit easier, when less information has to be remembered. For an example, a file *"My Thesis.doc"* found in the *Final Thesis* category on Figure-13 can be referred to with *"/University/Dejan/Final Thesis/My Thesis.doc"* or simply by *"Final Thesis/My Thesis.doc"*.

Paths are not paths as in hierarchy file systems; they are now views that represent intersections of relations' sets – categories. So far, the term filtering has been used instead of intersections, but both represent the logical AND operation. Paths are not important to the user. He does not interact with the computer or its file system thru paths. Paths are important solely to the overlaying applications, so that they can find their files. Although paths represent a narrowed view in the last category in it, which is of user importance for easier file locating, such approach is not needed by the computer itself. Computer needs only the filename and the last category in the path to uniquely allocate the file needed.

Path development and respective file listing development is illustrated on Figure-15. Note that the files listed in the final *"Peter/Work/Studying"* are already in *"Peter/"*. However, it is expected that each category has an uncountable number of files, and therefore, one can be very hard to scroll to, and moreover, to be spotted.

Conclusively, relational file system allows for better data organization that is user-oriented. It also allows a single file to "live" in different locations, without the need of actually copying it.

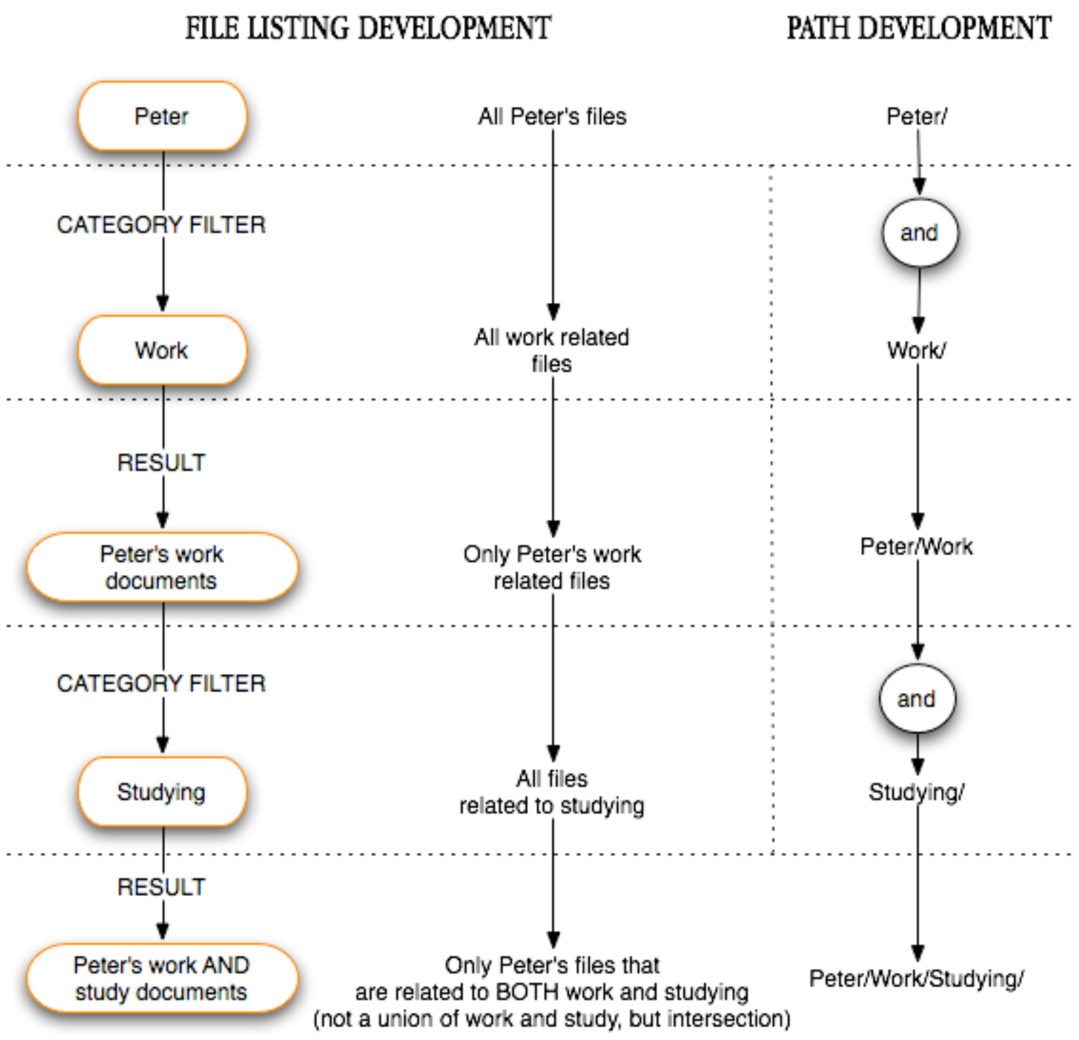

Figure 15 – File listing and path development

RFS Implementation Suggestions
— 4 —

Division of Information

RFS (Relational File System) assumes almost complete separation of data from its organizational information (relations). Those two can be separated even physically, and be stored on different physical mediums, while they are effective only in conjunction. Such approach will allow for operational parallelism. Therefore, RFS stores data in blocks in flat space, where each block has an absolute unique identification. These blocks' sizes are chosen as multiples of physical medium's sector size (default 512 bytes), and are preferably 4096 bytes (8 sectors), for normal, personal use. Available and taken blocks are indexed along with the relations, separated from the data. Category relations and multiple file belongings, as such, can be stored in a simple database, which should provide only trivial querying capabilities. However, this is not a necessity, as those simple relations can be implemented without a database. Despite, this document will assume that a simple database exists.

With block size of 4096 bytes and 4-byte wide block address (unique), one can address around 15 Terabytes of data, which is more than enough for personal computing. If that should change some day, it is only a matter system's preference. However, querying the relational database that holds the organizational and storage information, could be rather time consuming if a query is issued for each next or previous block (e.g. in R/W streams). Therefore, each block should hold next and previous blocks' IDs as offsets. Now, a user application can choose how it will access file, thru the database or in a stream. There is also another possibility left for the user applications to utilize: storage organization read ahead. An application can read a storage table of some file. By having it, it can easily jump back and forward thru a file, not skipping blocks sequentially as in double linked lists. Such approach is more than useful for multimedia applications, and yet it allows for some compatibility with current FAT file systems, that can be added later.

However, holding storage spread information along with user data is conceptually wrong, but this is the way to go currently, due to hardware limitations. Once mechanical (rotational) drives are out of use, data can be truly separated from the organizational and storage information.

Files

Every computer file system has to implement a file descriptor that holds file information. RFS file descriptor is held within the database in own table, and may look similar as Table-1.

File Descriptor Table	1. Unique File ID	2. File name	3. File type	4. File size (bytes)	5. First block address	6. Owner ID	7. Permissions	8. Creation date and time	9. Modification date and time	10. Modified by user	11. File comment	12. File color label	13. File meta data	14. Undo data????
1. (example)	0	File1.txt	Text	235	0	Root	600	7/06/2006 22:11	7/06/2006 22:11	Root	Test textual file	White	null	N/A
2. (example)	1	File2.bin	Binary	3220	1	Root	500	7/06/2006 22:14	7/06/2006 22:14	Root	Test binary file	Blue	null	N/A

Table 1 – File descriptor table

First column, File ID is the most important column in the file descriptor. Unlike popular file systems, as ones in Windows™ that identify files by the URL (address), RFS uses a unique number. Therefore, logical relations to a file are not broken if moved to another location on the storage device, or if the location changes name.

In order to keep track of free blocks and hence free space, absolute number of blocks available has to be known.

[Blocks] = [Physical Medium Size (bytes)] / [Block Size (bytes)]

[Blocks] = [Physical Medium Size (sectors)] / [Block Size (sectors)]

Now there has to be a data blocks table as on Table-2, with a single column of auto increment index that represents the Block ID, with a limit of a [Blocks] number. However, this might not be the best solution, as a bitmap use to track free blocks would take less space and be faster and therefore more efficient.

Data Blocks Table	1. Block ID
1.	0x0000000000000000
2.	0x0000000000000001

Table 2 – Data blocks table

A relation is needed that links file descriptors to the data blocks on the physical storage. This table is simple and may look like Table-3.

Block Links Table	1. File ID	2. Block ID	3. Order ID
1.	0	1	1
2.	0	0	0

Table 3 – Block links table

Order ID is necessary so that a correct data order (sequence) can be restored.

For multi-drive RFS, another table would be necessary, in order to track drives' spaces. This may look similar to Table-4.

Drives Table	1. Storage Device ID	2. Total Blocks	3. Used Blocks	4. Free Blocks	5. Bad Blocks
1.	IDEHDD1	10000	500	99500	0

Table 4 – Available drives table

Categorization of files

Users files need to be organized in a way understandable by the user, not only the operating system. This implicates that organization is done by meanings the files have to the user. Those meanings are nothing else than categories as discussed before. A single file can have multiple meanings, it can be view dependant and hence it can reside in many different categories. This is where hierarchical file systems fail. A single file cannot reside in different folders, unless copied. A shortcut can be created, but if the original file is moved, all shortcuts or links become unusable. A file is defined and found thru a view that consists of one or more categories. A file can exist in more views, since meanings have intersections, even though they do not have to. Deleting a file from a category (meaning) does not delete the file if the file exists in other category. Files can be observed as shortcuts to the data. Deleting the last file from the system actually deletes the file's data from the storage, unless of course, one insists on deleting all instances. This is all hidden from the user. The user can be tricked that he still may think he is working with files and folders as in hierarchical file system.

One might ask what is the difference, and hence the need for such a file system? The most obvious is the ability to search and navigate thru the files. It is easier to do a search on contained data within a set of files. It makes no sense to look for a textual match in binary files, from the operating system's point of view. This ability exists on today's systems, but cannot compare to the performances of a database. Yet, one does not have to index and re-index the storage device in order to keep the database accurate as modern operating systems do. The storage device's capacities are becoming larger every day. How will one index a 1TB disk drive that rapidly changes? It is obvious that the file system itself should be somewhat database driven.

The other ability that is not straightforward is the existence of dynamic categories. Categories are folder alike, but different in function. Categories are dynamical and there is no need to copy a file to the category as to a folder. Even though confusing, categories do not contain files. Categories are not structures. Categories are meanings that files can have. Giving a file a meaning of some category, the file can automatically show up in that category, in every view. A category sees a flat file system with no organizational structure at all, only pointing to files and other categories.

A category can relate other categories in almost the same manner in which a folder contains a sub folder. Therefore, we could say a category can have subcategories *in a view*, even though they do not "belong" to the parent category. Subcategories exist parallel with the category. They can reference each other, but not at the same time, as that would result in recursion. Moreover, they cannot relate forward to any backward parent, as that would result in cyclical structures. That is the natural approach, as it is in real life, verb dependant.

From the definitions above, one can notice that a category can be represented as a folder. It has sub-categories and lists files within current category. However, it has more than one category that point to it, as it has more than one category that itself points to. Note that the term sub-category should not be used, as there are no such parenting relations in RFS.

Categories can also be smart, and selective. That is, a category can select files from a set of files that fulfill a certain condition. An example of such a category is the one that will show, example given, all files created on January 1st, and contained the text "New Year" or have a META-Tag in common. This is accomplished thru SQL queries that can be assigned to categories. A user should be able to customize a category through a natural language that can be translated to a SQL query.

There are already smart folders on the Apple's Mac OS X 10.4 Tiger operating system. Apple has noticed the need for selective views. On contrary, those folders are very slow, since OS X uses indexing rather than central file system database. Opening a smart folder is same as typing in the keywords in the finding tool Spotlight. Difference between smart categories and smart folders will be in speed. Categories will list their files instantly, as they get their data from the database thru a query. Smart folders do a global search on the indexes spread over the system, every time user access them, which may last a couple of seconds.

A category should have own descriptor table. RFS implements such table as on Table-5:

Category Descriptor Table	1. Unique Category ID	2. Category name	3. Category Self Query string	5. Owner ID	6. Permissions	7. Creation date and time	8. Modification date and time	9. Modified by user	10. Category comment	12. Category color label	13. Category meta data
1. (example)	0	France	null	Root	600	7/06/2006 22:11	7/06/2006 22:11	Root	My France documents	White	null

Table 5 – Category descriptor table

Why a unique category name? Categories are as stated earlier, a meaning or an attribute in some context. It is like in real life; a word can have multiple meanings, dependable on the context of the sentence, but after all, it is a unique word. In the relational file system, a view defines the real meaning of the category (to the particular user).

Now that a category has been described, there is a need for category relation table, so that category relations can be implemented. This table may look like on Table-6.

Category Relation Table	1. Parent Category ID	2. Child Category ID	3. Creation date and time	4. Owner ID
1.	0	1	7/06/2006 22:11	Root

Table 6 – Category relation table

In order to have a browsing/navigating feature, there has to be a root just like in hierarchical file systems. As mentioned earlier, RFS implements a WORLD category with ID=0 that is not visible, but represents a start point or a root for browsing. This way, RFS will simulate hierarchical file systems, and will be compatible with all user applications. This solves one of the biggest concerns about compatibility with current operating systems, and adds more power to the file system. It becomes versatile.

To summarize, RFS has all the features of hierarchical file systems, yet it has more. In hierarchical file systems, each subdirectory has only one parent and infinite number of children; however, each category in RFS has infinite parent-like and infinite number of children-like relations. In addition, RFS can simulate hierarchical file systems, since current (hierarchical) systems are a subset of it. RFS enables for quick data searching as based on relational databases, yet it is naturally organized, since all the data is categorized upon creation, and searches are based primarily on categorization (attributes, meanings).

Conclusions
— 5 —

Personal computing, information technology and computer science have not reached a peak point as laics would say. Technology is still at its beginning, and true possibilities are yet to come. However, an obvious Research and Development (R&D) stagnation slows down its progress. There have been no real, inventive operating systems for decades [6]; most of the concepts have not changed since sketched at first. They just became fancy and stylish. This is a consequence of technology's mainstream use. Users are getting used to given, current solutions and resist radical changes, no matter how good and needed they are. Such behavior is of great decisive influence on IT companies, as new system's R&D becomes obsolete; current systems improvements are the primary goal. Project funds are divided in accordance with the same. Instead of having most funds in new product's R&D, its share becomes the smallest. Investments are only taking part because of the public image of that specific company. Moreover, this public image uses attributes as innovative, because of presenting vaporware prototypes on rare occasions. Such prototypes never arrive on our desktops, and as such do not matter. Only shipped products matter to the end users. However, this is not as bad as it sounds. There still are hard working fanatics, hobbyists that believe in further technological development. Having the large companies blindness of R&D, there is a sea of possibilities for small developers – venture capitalists and entrepreneurs. It is still possible that the next big thing will be just around the corner in neighbors garage by ambitious teenagers, again.

File systems of today do not satisfy users' needs and as Ph.D. Jakob Nielsen states in [1]:

> *"Emerging trends in user interfaces indicate that the basic file-system model is inadequate to fully satisfy the needs of new users, despite the flexibility of the underlying code and data structures."*

Suggested use of Relational File System and its virtual hierarchy, built on the fly, as core invention and contribution in this document, could be the basis for the file system of the future. Maybe one day we will be able to communicate directly with the file system, vocally, describing what we need. However, possibilities and usability of the concept should be researched more deeply, unavoidably with prototypes and usability tests.

This document tried describing that even foundations of modern computing need to be brainstormed again. Consequently, if foundations are subject to change, then everything else is too. This is a great exciting opportunity, that implicates new ideas, changes brought by others, or even better, brought by ourselves, as long as we keep our critical attitude towards technology. After all, its use is to serve us and make our lives easier, better and more productive.

Therefore, technology should adapt to our needs, not the other way around.

Resources

— 6 —

- [1] Nielsen, Jakob, "*The Death of File Systems,*" 1996, [Online] Available, <http://www.useit.com/papers/filedeath.html> (December 2005)
- [2] Gorter, Ohne, "*Database File System – An Alternative to Hierarchy Based File System*," University of Twente – Netherlands, August 2004, <http://ozy.student.utwente.nl> (January 2006)
- [3] Matthew, Thomas, "*When good interfaces go crufty,*" [Online] Available, <http://mpt.phrasewise.com/stories/storyReader$374> (January 2006)
- [4] Meyers, Jeremy, "*A Short History of the Computer,*" [Online] Available, <http://www.softlord.com/comp/> (January 2006)
- [5] Schwartz, Mathew, "The Interface Revolutionary," COMPUTERWORLD, 5 February 2001, (February 2006)
- [6] Pike, Rob, "*Systems Software Research is Irrelevant*," Bell Labs, Lucent Technologies, 21 February 2000, <http://cm.bell-labs.com/who/rob/utah2000.ps> (February 2006)
- [7] Tomitsch, Martin, "*Trends and Evolution of Window Interfaces*," University of Technology, Vienna, December 2003
- [8] King, Adrian, "Inside Windows 95," Microsoft Press, 1994
- [9] Jakob Nielsen on Usability and Web design - <http://www.useit.com/papers> (Multiple Papers and Essays)
- [10] Nielsen Norman Group – Interaction Design Solutions for the Real World - <http://www.asktog.com> (Multiple Papers and Essays)
- [11] The original Macintosh stories - <http://www.folklore.org> (Multiple Essays)
- [12] Home of Microsoft and the Windows operating system - <http://www.microsoft.com>
- [13] Home of Apple and Mac OS X operating system - <http://www.apple.com>
- [14] Home of the Linux operating system - <http://www.linux.com>
- [15] Don Norman's home page - <http://www.jnd.org>
- [16] Wikipedia - the free encyclopedia, "WinFS," <http://en.wikipedia.org/wiki/WinFS>
- [17] Wikipedia - the free encyclopedia, "Windows Vienna," <http://en.wikipedia.org/wiki/Windows_%22Vienna%22#Development>

Index of Figures

Figure 1 – The original 1984 Mac 7
Figure 2 – The 2006 iMac 7
Figure 3 – MacOS desktop and Control Panel 8
Figure 4 – MacOS file browser 8
Figure 5 – A Windows 3.11 Desktop 9
Figure 6 – The Windows 3.11 File Manager 9
Figure 7 – A Mac OS X desktop 9
Figure 8 – Root hierarchy 11
Figure 9 – Multiple relations 11
Figure 10 – A 3D mesh network 12
Figure 11 – Additional software for viewing files 18
Figure 12 – An example of simple relational structure 24
Figure 13 – State when starting browsing 25
Figure 14 – Forward browsing state 26
Figure 15 – File listing and path development 28

Index of Tables

Table 1 – File descriptor table 30
Table 2 – Data blocks table 31
Table 3 – Block links table 31
Table 4 – Available drives table 32
Table 5 – Category descriptor table 34
Table 6 – Category relation table 35