On the Internet, Network Computing, and Virtual Environments.

With The Abstract Units and Tunnels Primer

By José C. Elías

Updated with examples in 1998

NOTE: 99% of what's here was written during an all-nighter in 1993. The only changes made in 1998 where the introduction and the inclussion of more examples to make it easier for new readers to understand certain concepts.

Find out what the future of the Internet, and computing at large will be like in the coming future, and how a new breakthrough philosophy called "Abstract Units and Tunnels," for the first time discussed in this book by its visionary José C. Elías, will change the meaning and course of everything we've done in the past half a century of computing technology, and leaps ahead of concepts like Object-Orientation, Platform-Independence, Multi-Tier Architectures, and Open Systems.

With my deepest sincere dedication to all the people in the world who in one way or another made and continue making the computer revolution possible.

Chapters

Foreword / Introduction

When I first heard of computers as a kid, I saw them on movies and bank offices. At the time I came up with my own theory of how they worked based on my limited observations. I thought at the time that I could take any computer disk from any computer and insert it into any other computer, or take a program from any machine and run it in another, or even that we had no need to know where the programs were as long as they did what we wanted them to do.

I had no idea that computers would not work by simply turning them on, or that even in the same computer a program was not guaranteed to run unless it was running under the "right" circumstances (like a supported Operating System or specific graphics card or set of drivers). Although we've improved from the time when I discovered the hard truth (and my big disappointment), only since my discovery of the Internet and the coming of "The Web" have I come to see something that remotely resembles that original vision of mine. But we're still a long way off. Now more than ever we're dependent of a vendor's Central Processing Unit, or Operating System, or graphics library, or programming language, or hardware we can use. If you're not too technically-minded, you probably have asked yourself many times how come that you cannot run your friend's program on your computer, after all they both have a Central Processing Unit (CPU), memory, a screen, a keyboard, a storage device, etc., yet you can't, for some reason. And if you're technically-minded, and know what Object-Orientated Programming and/or Design is, you also know that regardless of all the promises and technical papers and conventions you have attended, in almost 100% of the cases the only way you can make two things work together in your computer is by making sure they're "compatible" with what you have and with each other. You cannot even use an Object-Oriented language like C++ and another Object-Oriented language like Delphi and make them easily coexist with each other without "hacking your way through" specially on different machines and operating environments. So, what will happen as in the coming years we continue writing larger and larger programs, as we continue leaving behind more and more "legacy code," as we continue outdating the latest hardware every two or so years, as we continue making everything larger and harder to manage? Are we approaching a state of chaos? How will we maintain millions of line of code? How will we transfer all this code to newer hardware? And will the new hardware support the previous applications? And if yes, for how long? This is exactly what I propose to offer a long-term solution for, this is what two fundamental concepts I've developed, Abstract Units and Tunnels will accomplish. At the same time as I present Abstract Units and Tunnels, I'll be discussing how we can improve things today and in the immediate coming future (including many predictions), so that we can make that childhood dream a reality.

I know some of the technical folks are now thinking (after reading the above paragraph) "Has he ever heard of Java, JavaBeans, CORBA, DCOM, ActiveX, etc. ?".

Be patient and stay with me, and you will see what is seriously wrong with all those architectures, and more important, how to solve the problem.

Don't be afraid if you're a non-technical person, a couch potato, a fully-certified computer guru, wizard, or hacker, a company CEO, CTO, or CFO, or an Internet search engine developer, this book is written in a way that will make sense to anyone with the most basic knowledge of what a computer is. My goal is to pass my ideas not just to a select few but to everyone, from the "Garage Engineer" to the CEO, from Financial Advisors to Philosophers, and to hopefully spark some ideas on at least one of you, since by personal experience with some close friends I know that a lot of people who probably have no idea what a computer is can contribute as much as a computer veteran, specially if you have a fresh mind who can look at things from a different angle than most people get used to after many years of doing the same thing. All it takes is some imagination, dedication, and most important, love and passion for what you do.

Chapter 0
A Brief History of the Internet

This first “chapter zero” I made it intentionally small because this book is not so much a book on the history of the computer industry as it is about its future. Furthermore this is not so much a chapter as it is a time line of major events in the making of the Internet. If you enjoy reading this chapter, I wholeheartedly recommend you read the book “Hackers - Heroes of the Computer Revolution” by Steven Levy.

I wrote most of this chapter zero already back in 1995 during a weekend, for a presentation I gave (as a student intern) at EMC Corporation in Hopkinton, Massachusetts, where I spoke about some of the topics in this book and even dedicated a whole part of the presentation talking solely about what Java would become (and did become a year later). That was a most gratifying experience which inspired me immensely to write this book.

Before I continue I should point out that the remaining of this chapter is purely for your “educational edification”, and skipping it will not affect you on the reading of the other chapters, however if I were you I’d read it, and consider it a crash “Internet History 101” course. Some of you Internet veterans my find one or two things in here you didn’t know.

The Internet, contrary to popular belief in non-computer circles, is actually nothing new, as a matter of fact many of us were not even around during the Internet beginnings back in 1962 when Paul Baran first described packet-switching networks for the first time. Note that this early, there was not even a name for what today became the Internet.

Seven years later, in 1969, the Department of Defence started founding the Advanced Research Project Agency Network (ARPANET) which would later evolve into today’s Internet. At this point in time the University of California at Los Angeles (UCLA) became the very first node in the network.

In 1971 (the year I was born) there were 23 hosts, and in 1973 ARPANET established connections with England and Norway. This is also the year that Ray Tomlinson invented “Email for Distributed Networks.” Email immediately became the “Killer App” for the small network.

In 1974 Robert Metcalfe’s Harvard PhD thesis outlined Ethernet, and in the same year Vincent Cerf and Bob Kahn detailed the Transmission Control Protocol (TCP) as a protocol for packet networks. At the same time the first commercial version of ARPANET was opened by BBN. By now there were 62 hosts.

In 1976 Unix-to-Unix Copy (UUCP) was developed at AT&T Bell Labs, which led Tom Truscott and Steve Bellovin to establish Usenet (a distributed news system based on the concept of "discussion newsgroups") between Duke University and the University of North Carolina (UNC).

In 1982, TCP becomes the standard ARPANET protocol, and in this same year the word "Internet" was beginning to be used, meaning "INTERconnected NETworks." Also this year the European UNIX Network (EUnet) began operating. There were now 235 hosts.

At 12:01 AM of January 1st 1983, everyone switches from the old Network Control Protocol (NCP) to TCP/IP (the IP means Internet Protocol) as the primary protocol for the Internet. Also, the University of Wisconsin developed "Name Servers" (which today map numeric addresses to human-friendly name aliases) and Berkeley released UNIX 4.2, incorporating TCP/IP. These changes made the Internet grow to 500 hosts that same year.

In 1984, a formal Domain Name Server (DNS) system was introduced, and the Japan UNIX Network (JUNET) was added to the Internet. Also, a new Network News Transfer Protocol (NNTP) enhanced Usenet over TCP/IP. This same year was the beginning of the historical 56 kilobits/second "backbone" added to the Internet in order to connect 5 supercomputing centers (note that most *single* users today can order a 56kbps line from their local Internet Service Provider (ISP)). What followed after that was an explosion of connections reaching 5,000 that same year.

In 1987, the National Science Foundation (NSF) handles over the job of managing the "NSFNET" to Merit Network. Soon, Merit, IBM, and MCI found Advanced Network & Services (ANS). The Internet is now 20,000 hosts and growing.

In 1989 Ohio State University establishes a relay between the Internet and CompuServe, also the NFSNET backbone gets a major upgrade to "T1", thus moving from 56 Kilobits/second to 1.544 Megabits/second. This year the Internet celebrated its 100,000th host.

In 1990 ARPANET ceases to exist, and Mitch Kapor founds the historical Electronic Frontier Foundation.

In 1991, Thinking Machine releases WAIS (Wide-Area Information Service) and the University of Minnesota demonstrates Gopher. These two could be considered at the time a "sneak preview" of what was happening to the Internet. At the same time, the US government establishes the National Research and Educational Network (NREN). This same year the number of hosts was 376,000 by January, and 617,000 by October.

In 1992, CERN (The European Laboratory for Particle Physics) releases WWW (The World-Wide Web), they becoming the first WWW address on "The Web", and the Internet is by now 1,000,000 hosts strong. Note that back then, the WWW was still a non-

graphical means of navigating "Hyperlinks." Soon after the text-based Lynx "Web Browser" appeared. This first year the first Mbone multicasts, giving us a hint of a future multimedia-rich Internet. This year the Internet backbone is upgraded again, this time to T3 (44.736 Mbps). [PLACE THE NAME of WWW inventor in this paragraph!!!]

In 1993, the White House comes online, as a sign of things to come. This same year businesses discover the Internet and the Internet starts getting media-hot. Hype grows.

In 1994, on the 25th anniversary of the Internet, Mosaic (a project headed by a then-student Marc Andreessen) is released by the National Center for Supercomputing Applications (NCSA) as a graphical interface to the WWW, taking the Internet completely by storm and causing the WWW to start growing exponentially. More than 3 million hosts are all of a sudden on the Internet.

In 1995, the National Science Foundation stopped funding the NFSnet, thus handling the Internet to commercial organizations. This produced more growth, from about 4 million hosts in January to about 7 million in August. In the meanwhile, the National Science Foundation started founding the vBNS (the Very-High-Speed Backbone Network Service), running in the Gigabits/second range, and giving us a glimpse of what's to come. Also Netscape is founded and SUN releases the Java "language for the web" publicly for the first time, giving us a peek at platform-independent executable content over the web. At the same time, the first notions of an "Internet Appliance" start popping up in the minds of some industry analysts. The web also sees a lot of audio/video applications beginning to emerge, in the areas of real-time audio broadcasts, real-time video streaming, and telephony.

In 1996 Java becomes the technology hit of the year (if not the biggest thing since the WWW), gaining support from every major Operating System vendor, software and hardware developers, and even chip manufacturers. Microsoft quickly joins the Internet wave after having been caught off-guard, deciding to "webtisize" their complete family of Windows products. The term "Intranet" becomes a common noun in computer circles, meaning an internal network based on standard Internet/WWW technologies. Also the now-official "Network Computer" popularized by Oracle's CEO Larry Ellison creates a lot of controversy, denial, and backers. At the same time, the US government is (at the time of this writing) planning on how to tax the emerging Internet Netconomics system. By January1996 the number of hosts was around 9.5 million.

In the first quarter of 1997, there were 50.6 million Internet users in the USA alone, and by the end of 1997 there were over 100 Million users on the Web, over 350 Million web pages, and analysts predict business on the Internet to become a 3 Trillion industry by the year 2002.

[NEED MORE GLOBAL DATA HERE!!!]

Chapter 1 [MOVE THIS to a later chapter!!!]
Beginnings of Virtual Worlds

BRAINDUMPS: [might take this chapter out???]

TOPICS: I've always told my friends that games have been the driving force behind most great new consumer technology (look at Virtual Reality and 3D games today, plus at the future of VRML).

- The Chess game hack at MIT back in the whereabouts of 1961, a TX-0 and a Digital Equipment Corporation PDP-1
- Quantum Link (later America On-Line) and LucasArts's "Habitat" on the Commodore 64: the first graphical distributed virtual environment.
- 1983's Neuromancer captivates the imagination of hackers all over the world. The video game comes out later.
- 1993's SEGA's Virtua Fighter
- 1996's Quake (Virtual Worlds across multiple systems)

According to popular theory, back in the whereabouts of 1961, a group of hackers wired two machines (a TX-0 and a PDP-1) together to make their professor believe that one of the machines was playing chess intelligently, while in reality another guy was making the moves in the back room on the other computer. This was not probably the first time such a connection was established, but it did bring up an interesting theory of mine: most new great technology advances happen because people want to have fun.

Several years later, in the late eighties, a group at LucasArts created something amazing. By using Commodore 64 computers, a simple 300-baud modem, and a central network of computers running on a service then called "Quantum Link" (today, America On Line) [NEED TO VERIFY THIS!!!], the LucasArts engineers where able to create the first consumer-wide version of a visually virtual environment where each user assumed a character on the screen and met other people who in reality where physically located hundreds or even thousands of miles away. The product was called "Habitat", and it changed a lot of people's lives. For the first time people were not just typing commands and seeing text descriptions (or text-based graphics as in the UNIX-based Multi-User Dungeons "MUDs"), but actually "seeing" other people, interacting, and sharing experiences with them. The system was several years ahead of its time and it was shut down eventually and the technology sold to a Japanese technology company [RESEARCH name of this company!!!], but the impression had already been engraved in the minds of visionaries.

In 1993, at about the same time the White House joined the Internet, SEGA released a game that forever changed the perception of how end-users visualize virtual environments. The game was "Virtua Fighter", using the same engine as its graphics successor "Virtua Racing". It was at after this time that PC games designers realized the

need to improve on the home experience, and many influential titles came out, like Wing Commander and Myst, and the highly popular Wolfstein 3D.

In the meanwhile a lot of companies were hard at work on the Internet designing "Avatars"-based centers. An Avatar is an identity you assume under a virtual world. However there was never a way for a user to walk from world to world as different developers created their own proprietary. However, in 1996, ID software released a successor to Doom (which was the successor to Wolfstein 3D) called Quake which allowed users to engage interactively with other "Quakers" from anywhere in the world. They had not done anything completely original, but they did bring the concept of Virtual Environments to a large portion of the population.

The next big evolutionary change will involve allowing users from completely different environment to "carry" with them their identities as they wonder through the Internet with their Avatars.

Blah blah blah…

Chapter 2
The Evolution of the Computer in the Internet

BRAINDUMPS:

TOPICS: Discuss here the how computers went from single machines (in parallel in the "PC" world and the "Corporate" world) to how they started getting connected. How non-expandable technologies died out while "open" technologies survived (IBM's AS/300(???)) is a great example; { talk about "code upward compatibility" and "third-party clones })

The logical evolution of computing:

- Hardwired machines (ENIAC)
- 1 machine, proprietary, own machine language (first machines)
- 1 machine, open to 3rd parties (IBM's mainframes)
- 1 machine, with higher-level language (Apple II, C64)
- Many machines with higher-level languages plus custom-extensions, but no binary compatibility (PCs and Macs)
- Many similar machines with distributed work (but frequently must share the same architecture)
- X-terminals and "dumb" (VT100-like) as front-end to a server
- Many workstations using one server
- Many dissimilar machines, networked together using agreed-upon messaging protocols, but cannot execute other's programs
- Many machines, all using a common "virtual machine", thus enabling both data communications and processing, smart and dumb terminals, and platform-independent distributed applications.
- One big global "Virtual Network" (VN), with virtual machines that runs anything, communicate with anything, and also take advantage of each other's advantages. At the same time has no central authority to avoid the whole "web" from collapsing (point to Netcom "Ampersand Collapse" of 400,000 people for 12 hours in June 1996, and ISP routing table error which caused millions of people to be disconnected for several hours in April 1997), and at the same time allowing for individual (either a person, group, organization, or company) improvement of the network. The whole network is alive, with improvements being made constantly without restraining its operation, becoming smarter and more useful every passing moment.

Chapter 3
The Metamorphosis and Future of Intranets

BRAINDUMPS:

TOPICS: Intranets will be extended from corporate buildings to the home, the neighborhood, the town, and the city.
Intranets will be physical (like inside a house, or town) or virtual (like a group or organization).
Intranet content will be private (like today's company's firewalls) or completely or partially public, and other variations (like guest, or limited visibility, or buying access rights, etc..)

In today's increasingly-electronic business world, the Intranet has become the biggest of all the buzzwords and the focus of multibillion dollar investments and companies struggle to catch up to Internet-based technologies as the basis to compete in the new networked world upon us.

An Intranet is just the act of taking all the technologies developed for the Internet (like traffic routers, networking protocols, web-based applications, etc.), and using them inside a company's network for its own purposes. The rationale behind it is that instead of a company having to develop its own technologies or using third-party proprietary technologies as the basis of its network and its business, why not use readily-available open Internet technologies which offer everything you could ever come up by yourself, plus much more? This begins to make even more sense when you realize that you can capitalize of the millions of man-hours of work that are put every month by thousands of companies whose only job is to make everything better on the Internet. In essence you have the whole world working to solve your problems.

There's also a variation of the Internet and the Intranet called an "Extranet." An Extranet is nothing more than the act of two or more companies (or departments within the same companies) using the Internet as the communications medium between their Intranets so that they could in effect do business between them on the Internet, transparent to all Internet users.

However, in time as the reality of physical space sinks into the use of the Web, other forms of these "Hybrid Internets" will emerge. For a simple reason consider calling a Taxi. In almost every case you're going to request service locally, so it makes sense to optimize the speed and quality of service of Taxi web sites within local areas. This does not mean restricting access to other places, as for example a person might want to order a Taxi for someone else.

These new Hybrid Internets will be categorized by three distinct attributes

- Geographical Area
- Physical or Virtual (or a mix)
- Public or Private (or a mix)

Geography will play an important role in the Internet of the future, as towns, cities, an even entire states and countries try to offer the best possible Internet experience to users. Politicians will undoubtedly use this as a mayor force to gain votes. Also imagine landlords making a big deal of why their buildings are the best to live in because of their fast access to the Internet from every room. Expect to see signs as you go into cities saying things like "The City Wired for the Future" or "The Fastest Internet City in America", etc. This phenomenon is already occurring with a whole country, Japan, where every home will be wired with high-speed digital cable within a decade. It is also occurring in India and Malaysia, where the government is sponsoring "Technology Cities" in order to attract a global community of high-tech investors. [GET NAMES OF CITIES HERE!!!]

However, a different type of Hybrid Internets will not be bounded by physical limitations, as they could be Virtual Networks. A Virtual Network is one which appears to anyone as one physical network, even though the actual elements that compose the network could be in the most distant parts of the planet. A good example of a Virtual Network is one made for an International Sports Club. In such Virtual Network every participant feels like if they're all "behind the same walls," even though each person is really far away from each other. Another example of a Virtual Network is a Global Corporation Network, which can use a Virtual Network approach to make all its branches seem like if they are all "under one roof." A small-scale example could be a Family Virtual Network, where the whole family shares work, experiences, and keeps in touch, even tough some of the family members might be located in far away parts of the world.

One issue that will become important, regardless of the type of Hybrid Internets we might come across, is the issue of privacy and of access. For example, you might not want every Internet user to have access to your Family Virtual Network, but you might want to allow some close friends to do have access. Likewise, a fresh bread store might have buying rights for locals only, and a whole city might consider anyone contacting the City Net a Virtual Visitor or a Virtual Guest as opposed to a resident.

The security and access approach has two objectives. One is to obviously keep things that are supposed to be private as private, but also to allow for information to be conveniently customized to non-natives (i.e.: their language, customs, habits, payment methods, etc.)

Chapter 4
From Software Suites to Inter-operable Components

BRAINDUMPS:

The applications running in the tomorrow's computing devices will shift from same-developer product "suites" to multi-vendor inter-operable software components. This will allow for the corporation or home user to buy what it considers best for its goals out of each software category out of all the vendors. You won't be stuck with one or more components of a software "suite" becoming obsolete in relation to other vendor's, and in effect what it's going to be achieved is that the "suite" will be assembled *by the corporation or home user* as opposed to by the provider. Note that this is not "*going to what we had before suites years ago*," because before, those individual software pieces could no blend with each other seamlessly the way it will happen now. In other words, disparate applications will seem to become one application, but consisting of individual behind-the-door modules, each of which will be replaceable at any time dynamically and by any other vendor's superior technology or better customer support or any other criteria that the user considers makes a change worth. Also talk about the downsizing of software makers as people have more choice. The importance of Component s-based Systems Engineering.

Chapter 5

The Modular Computing Platform

(basically taking the ideas from the previous chapter which apply to software and applying them to hardware)

BRAINDUMPS:

TOPICS: Extending modular components to the CPU and OS, plus fighting "PC Obsolescence" (modular PC, mention Panda project!!!), also talk about a data-independent OS, this way people can get to choose the best OS for their needs in a manner that is completely application-independent (so we can take advantage of new fast-response OS's like "Be!" or free OS's like Linux, or even Extending modular components to the CPU and OS, whatever lays at the horizon (Plan 9 at Lucent?). Talk about how the User Interface (presentation layer) will be replaced by a Web Browser-like interface (or anything else that will make managing the user's data easier). Also hint about a way for all applications to talk to each other (discussed in a later chapter). **The whole point of this chapter:** Almost everything can (and should) be modularized and "plugged in" dynamically or "swapped" and make it work with what already exists (give example of user who bought a PC with a slow bus speed and now wants to use a faster bus for video editing and a new OS, but also keep all his data (and access to it) plus all his hardware components (graphics card, removable storage device, DVD, network card, etc.,) and also talk about how this is somehow done today with modems that can be used with any serial computer, and the Mac and Windows 95 "Plug & Play" mechanism, also how Panda computers can even upgrade their bus structure in a module) Also talk about how the technology behind HotJava is the perfect example of a concept that can be used to dynamically update the "OS" used to access your data, and how the most popular browsers (like Navigator and Explorer) will eventually naturally go towards this same direction (either as self-updates or auto-updated by other modules (like Castanet)).

Chapter 6
Nomadic User Interfaces

BRAINDUMPS:

What I refer to as a Nomadic User Interface (or NUI, I pronounce it "New E") is a user interface that (thanks to many of the concepts I discuss in this book) allows it to "travel" with the user. A Web Browser with access to all your private and non-private data is an example of a very simplistic Network-based NUI.

A NUI can travel with or to the user in two different ways. Either in a Smart Card or via the Network. The way it'd work in a Smart Card environment is that the user whenever he uses a computing device at home, always has inserted into a slot his or her Smart Card (or maybe I should call it "Identity Card"). So the card (which most likely will be in a solid-state form) will save the minimum required "settings" as the user works, so that if the user later decides to move to another computing device all the user has to do is insert the Smart Card into its slot and automatically the NUI (and all his work) will appear on the screen (or a holographic display or whatever the user interface is, maybe even completely voice-driven) exactly the same way it was left back at home. The second way of achieving this does not require a Smart Card, instead all the data is saved into the Network, and when the user goes to a remotes place physically he or she "logs in" much like in today's systems by identifying himself (which could mean anything from finger-printing recognition, to body-heat pattern-recognition systems, or retina-recognition systems, or voice-recognition systems, or plain passwords for not-to-secure things) and again just like before the NUI will appear right there again in the same state left previously (a traditional Web Browser could be modified for this functionality).

What this means is that it will become suddenly practical to offer these "Network Terminals" in every imaginable place, and offered to people upon request. This includes airplanes, trains, busses, restaurants, hotels, and other any place where it is important for one reason or another to be "connected." We'll probably see "Network Terminal Booths" just like we see telephone booths today where can go in and quickly do some work (of course, a user will always have the option of a wireless version of this Network Terminal which will communicate using encrypted communications with the nearest "Cell Station" which can be located inside a hotel, on street lamps, or even in the middle of the desert).

It is very important to note however, that the real "trick" to making these Network Terminal ubiquitous is in setting standards that will allow any Network Terminal deal with any type of user data (either information *browsing* or *processing*), and thus ensuring that whatever you plug into will deliver what you expect.

Note that a major advantage a network computing device coupled with a Nomadic User Interface has over existing computing schemes is that it offers almost complete protection of the user's data and working environments. So if for example you NC stops

working all of a sudden and you're doing something critical, you can always get to *any* other terminal and resume work. And since your data resides in the Network, it is very likely that your network space provider will have sophisticated data mirroring techniques that will ensure your data is actually safer with them than in today's hard drives. This does not mean that we'll stop using local storage, but it does mean that depending on what type of data we have we'll be using either the Network or local removable storage (things like home movies you might want to store locally, but things concerning work will probably be stored in the network so that you can access them from anywhere anytime).

Chapter 7
Personal Network Servers

BRAINDUMPS:

Talk about Personal Network Servers as well, and why we need as big a pipe coming *out of* the house as opposed to *into* the house only.

I estimate that 60 to 80% of all the computing power in the world is never used. For how many hours a day does the typical user of the 50+ million PCs use his or her PC? Probably 1-4 hours, usually after getting back home from work. So why not put this power to use? We could develop a framework where for example, the scientific community would pay regular PC users (or ever corporations) to use their PCs for computing power during the hours their computing devices would normally be idle. This means that a better network infrastructure than the one we currently have today would need to be in place to really take advantage of this, which means that Asymmetric Digital Subscriber Line (ADSL) would best be replaced by a symmetric approach (where the amount of data going into the house is the same as the one leaving the house, as opposed to ADSL where the data bandwidth out of the house is usually much smaller than the bandwidth coming in), or even better, an approach that dynamically self-adapts to bandwidth demands.

This scheme would benefit users and the "Service Requesters" (as opposed to Service Providers) in general. Imagine running a computational-intensive application like weather forecasting or a model of the beginning of the universe, or even the interactive forces in a small quantum mechanical system by distributing all the work among hundreds, thousands, or even millions of processing units in parallel. The benefits would be enormous. This scheme would also prove popular among many users since they'd get paid for doing virtually nothing. Who knows, maybe one day we'll see local companies providing free Internet access in exchange of computing power, or universities providing a tuition discount for students willing to allow the research faculty to use their home PCs overnight. The possibilities and amount of innovative new services this scheme could create are essentially limitless. One such new innovative service would be the creation of companies whose only job is to become "trusted" in the mind of the home PC owners, and to essentially become an intermediary between a home user and a corporation looking for willing users to host their computational applets. A service like this would be important, if not essential for users who don't trust every company out there to run their applets on a user's PC. In other words, the user trusts such an entity as having done a background investigation on the company who will use the user's PC. This in turn, will create even more services even outside the direct computer industry. For example, insurance companies could offer a service for the above-mentioned Intermediary Trusted Third Parties (ITTPs ???) where they would financially protect them in the even that a

corporation accidentally (or purposely) destroys resources on a user's PC via their applets (this is similar to how hospitals get insured in the event that they get sued by patients).

Chapter 8
Abstract Units and Tunnels

BRAINDUMPS:

The next step after Object-Orientation: Abstract Units and Tunnels, which provide for the notion of "Abstract Units" that can be either software or hardware (the decentralized "system" or "network" has no knowledge of the Abstract Unit being software or hardware, or anything else for that matter) and that communicate with each other via "Tunnels" which are a way for any Abstract Unit to communicate with any other one. Note that the notion of Object-Orientation is not lost in all this, it is just encapsulated as a subset of Abstract Units, which in effect are taking the principle of Object Orientation to a higher abstract level where everything is an Abstract Unit. Java is a good example of something that resembles part of this scenario.

Abstract Units can be anything from raw data, to an algorithm, a CPU, a programming language, a communications protocol, a database system (which as an Abstract Unit manages other Abstract Units, which in turn can be anything!!!), a user interface (which like the database example above can present and manipulate any Abstract Unit, or it can blend with other Abstract Units), to complete computing systems, to "Abstract Unit Handlers" (the equivalent of today's Operating Systems) but which can handle any Abstract Unit (while being an Abstract Unit itself), etc. The advantage of this is that anything works with anything else, so you can use the User Interface you like best from a variety of versions from different manufacturers to manage your data (so for example you can use an MS-DOS-like prompt, or a Mac-like interface, or a browser-like interface, a voice interface, or a VRML-like interface, or anything else or a combination of all of them to manage your same data; you'd just pick the interface you consider best for each data type), and mix and match any database, language, graphics card, or graphics program, or storage device, or Network Access Provider, or CPU type, and make it all work into your own system, just the way you want it to be.

This means that grandma will probably have something that has an interface which is mostly speech-based, and with some huge-size fonts, and easy one-step (or one voice-command or anything else) access to voice and video conversations or assistance, while an Abstract Unit developer might have a super-fast system with a flashy virtual environment interface and the latest natural and visual languages and tools.

What all this means is a complete departure from most of what we've gotten used to in the past half a century, affecting everything from the way we design CPUs, to the way we design languages, networks, and peripherals. But how will all this be accomplished in the first place? Well, although it might seem impossible at first, all it takes is to follow simple philosophical rules, much like the way we humans communicate with each other by following the one rule of using a common language to talk to each other, or an interpreter or translator to talk to someone foreign. These "rules" are actually better defined as an "agreement" or "standards" between the industry designers. These

changes would make everything we have today still possible, yet provide us with things most people cannot even imagine today. **In other words, since everything is an Abstract Unit, it means that if each vendor decides to, they can all have their own protocol, their own set of graphics libraries, their own languages, and yet coexist with each other**. If this sounds impossible, think about how humans behave and interact in the real world: each household has a different TV set, a different car, different clothes, hobbies, opinions, feelings, thoughts, and do different actions during different times in their lives, and yet they all get to watch TV, to drive the car, to do what they want, and most fundamental of all, interact with each other. So, if you stop thinking about the details of how we're going to manage to make my User Interface work with your data, or someone else's, or how someone's new CPU design will help anyone use its computational capabilities, and instead concentrate on what is it that we're really trying to do, or in other words, what is the fundamental element behind all this, we'll not only discover that it is doable, but that it is also not that hard at all. So, how will all this work? Well, to get you started, think about how not long ago you had to have a Mac to run Mac programs, a PC to run PC programs, and a vendor-specific UNIX box to run that vendor's UNIX program, and yet in the past recent years, after the release of the World Wide Web (WWW) we're now able to use information stored virtually anywhere, from virtually any platform, and more recently we can even run Java programs on virtually any platforms as well. There's even connectivity options in technologies such as the Common Object Request Broker Architecture (CORBA) which allow almost any system to talk to almost any other system. So, in this example, what is the fundamental "thing" we're trying to deal with? (hint: it is not the web, CORBA, or even Java itself), the answer is simple: **all we're doing is trying to access and manipulate data**, nothing else (sounds a lot like the Object-Oriented principle of Object and Methods that operate on these objects, doesn't it?, in other words, consider what you see on a web page: the data as an object, and the web browser as a method that operates on the data (i.e.: on the object)). Well, to even get closer to where we're heading, have you noticed that in this web-centric view of "data", if you're accessing the data from a PC you're most likely interacting with this data on a computer using an Intel-derived CPU, and at the same time a graphics card from one of dozens (or hundreds) of manufacturers? And if you're on a Mac you're using a Motorola-family processor, and the if you're under UNIX you can have a variety of high-performance processors as well? Well, stop and think for a second: If this thing we call "The Web" keeps expanding (and I think you know the answer to that), very likely will also its "data" and its applications used to interact with this data. So, if you get to a point where all you're doing is interacting with this data, from your platform, but the same holds from your neighbor next door, then, why should you get stuck buying a system that uses an expensive CPU when you can just go get an equivalent (or more powerful) CPU for less money? This is a philosophical view which applies not only to CPUs, but to anything being hardware or software (even life itself).

We've seen this before with the big "PC" price wars, where a "PC system" that performs the same (or better) as another "PC system" is sold more inexpensively. But what will happen when all we care about is the data we want to manipulate and how to manipulate it and not what we use to manipulate it? It means that the "PC price wars" will extend not only to the internals of what today we call a PC, but to the externals as

well. This means users will buy what meets their equation for cost-effective or power-effective or whatever their criteria is (like customer service, reliability, easy of use, etc), and will decide on that. What this means is that if company "A" makes a CPU or an OS that millions of computers users use, but company "B" just came out with a CPU and/or OS that does the same, at a lower cost, or with features I like better, then I'll go for "B" (since I will *still* be able to access and manipulate my data!). Today that choice is hard to make, since we're entrenched in a world of complete proprietary "standards" like Intel CPUs and Microsoft software (I have nothing against these companies since after all they each have a duty of maximizing their shareholders' profits (plus they where the ones to bring computing to society the way we have it) but I hope they each don't have the erroneous belief that they can continue doing this forever in such a rapidly-changing, dynamic, and open-systems world).

The coming technologies will need not only accommodate what's available, but what's to come as well. In other words, **a completely dynamic and self-adapting system much like life itself**. This might sound like science fiction but if you had the chance to try out Sun Microsystems's HotJava web browser, then you were exposed at a little piece of the future. What HotJava can do is to allow the Java-driven browser to adapt to new technologies by self-adapting itself to new and unknown "data" (actually it is not as straightforward and powerful as it sounds, but they are heading in the right direction) and installing a new "data handler" for it. So if for example the browser comes across an unknown video or teleconferencing data type, the data provider can also include the necessary platform-neutral module that will "attach" to the browser dynamically and manipulate the data. In other words, the browser of the future will "learn" from this "experience" much like we do, and next time it comes across similar data it will know how to handle it. It should probably be useful to include a "forgetful" mode into the software so that if it doesn't see the data type for a long time it will disassociate from it in order to for example clean up resources used to support that data type (which can be a local or remote data manipulator (a "skill") or an address to where to get that data manipulator from (a "reminder"). One thing to keep in mind though, I use the term "browser" not meaning just today's web browsers, but the way we'll be handling data in the future (where data can even come with its own "browser" tailored to its purpose).

However, I have not yet answered explicitly the question of how all this "Unit Abstraction" be accomplished (although implicitly I more or less have, at least with the HotJava example above). The solution is composed of two parts. The first part is to create "Tunnels" which are connections analogous to "sockets" or "Remote Method Invocation (RMI)"or "Application Programming Interfaces (APIs)" or CORBA-based systems. The function of a Tunnel is to channel messages between Abstract Units (either asynchronously or synchronously). Any Abstract Unit can send messages to any other Abstract Unit, but it is not always guaranteed to get a reply (this is still pondering a question in my head, probably a better solution is to do reply with a "Cannot Do That" message, and also account for the possibility in time of not ever getting a reply). In other words, it makes no sense for an Abstract Unit pointing device to send information to a storage device, so the storage device can simply ignore the message or send back a message saying "I don't do that" (this does not mean however that for some bizarre

reason a hard drive manufacturer might decide to do something with a joystick or mouse, and then do something about these messages (this is exactly where schemes like this lead to innovation)). The point is that such cases will almost never occur since the programmer knows that the Abstract Unit pointing device will be talking to an Abstract Unit User Interface and/or display or application of some sort. The second part to the solution is an Abstract Unit Virtual Execution Machine, which today's best example is Java. So in other words the Virtual Machine is used to interact with the data, and the Tunnels are used to make sure all Abstract Units can talk to each other (so think of this as a super-component architecture where any "component" (or actually, Abstract Unit) written or built by anyone else can interact with any one else's (the market itself will define Tunnel Types to talk to as many other types as possible, and the user will decide which Abstract Units to integrate with which other Abstract Units (or the whole thing will occur dynamically via Abstract Unit Software Agent Brokers)). In other words, a "clock" Abstract Unit might reply to Tunnels requests from a variety of Abstract Units like a Word Processor program that wants to embed it inside a document, or a database that wants to know what time it is, or an electronic piano for timing purposes, etc. while the user has the chance of picking which clock to use to embed or to reply to the database (since everything inter-operates). Also note that the underlying Virtual Machine should be an Abstract Unit itself, **giving it the capability of "learning" new instructions, and implementing them in either software or hardware** (like on a programmable DSP). **Eventually, as new Networking Appliances come out, they will incorporate the most popular that the majority of the VMs have "learned" into actual hardware, making them much faster than their software-only equivalents. The whole process will cycle itself much like "the market" works in a capitalist society, all by itself, apparently out of control, but in reality in perfect harmony since everyone (in this case, every device) will just be trying its best to do their best (much like humans try their best to do their best).**

This scheme has very deep philosophical ramifications, extending everywhere to everything like human behavior and learning (psychology), genetics, society evolution, international relations, diplomacy, politics, and of course philosophy. We will be confronted with a system that will evolve and self-adapt to the times. Old technologies that nobody uses will dynamically cease to exist, and new ones will dynamically spread to everyone who uses them, and even then, new technologies are not guaranteed a long life unless a sufficient amount of users supports them, much like in a democracy. There will also be evolution of technologies, like breeding a popular video player with a popular audio player to produce a movie player, but then such marriages might have already occur, and competition will occur between all these movie players, and success again determined by which one adapts best to the needs and wants of the marketplace, in the true spirit of pure Capitalism, while still [for example] allowing users to see movies decently (with a lot of the new functions running in software only) in their movie players (and again, they could be "up to date" by adding a small plug-in hardware module that makes their system perform better).

The possibilities for such a system are boundless and beyond the imagination of today's society, but it will certainly work out for the best as it becomes the next deep

fundamental change in the course of humanity, much like the Internet did at the end of this century. [MOVE THIS (or copy/rephrase) to afterword section on Abstract Units and Tunnels!!!]

Afterword

147

Index

Ideas to incorporate in the body of the book

PC makers are happy where they are, since it's good for them for their products to become obsolete as soon as they arrive into a household (imagine investing $2500 every three years on a new system, plus all the hardware and software upgrades). So it'll probably take a company that wants to compete with the PC makers (maybe a current "underdog" PC maker who can expand quickly and become a leader, or a company who wants to expand from consumer electronics into home computing appliances) in order to make the Network Computer dream fly. However, maybe these companies don't realize that if they provided added value to a current system with a new add-on, people will still spend the money in new equipment, but at least no because they were forced to but because they want to (as would be the case of a new ultra-realistic 3D graphics card that noticeably enhances the immersion experience, or a new add-on card that provides enough power to understand natural language from anyone in the home without training, or a new low-cost interface that allows people to use their home Camcorders to do real-time video communications, etc.). Good examples today in hardware are the "Snappy" video frame grabber by Play Inc., and Kai's Power Goo, both of which provide sufficiently "fun" value to users that they buy it just for the experience. Now, if you take into account that you can provide a network computing appliance for a modest price a family who otherwise would not have bought a PC can now afford, and you then do the math on the potential market for these devices, and then multiply all this extra market you have and all the "add-ons" you can provide over the years, I think any PC maker should think twice before saying no to the idea of a Network Computer. One important point to make here is, that the idea of a NC is so doable, that if the big PC companies don't do it someone else will (that reminds me of the old saying: "Evolve or Die").

If the NC had not been envisioned today, the PC would then have evolved into a NC (and probably will) anyway in order to survive in a new world of consumers who will grow more and more accustomed to just "plugging in and go," just like their toasters, their TVs, or their cars. As a matter of fact, the PC of the future will be a modular, reconfigurable, upgradeable, and self-adapting computing device, or in other words, a Network Computer as described more or less today. What all this means is that we *will* get to see the vision of a Network Computer, it's just a matter of when and how we get there, either by the short route, which is to start making NCs now, or the long one, which will occur naturally as the PC evolves (or dies).

RSVP, The Resource Packet-Reservation Protocol, will change the way data migrates in the Internet. What RSVP essentially allows users to do is to reserve bandwidth before hand by paying for it. This will migrate the current "socialist" Internet to a more "capitalist" Internet where you get what you pay for. However, this does not mean that the current protocols will disappear, but it does mean that users will be able to pay for quality of service, so as to be able to watch a "Webcast" live, made possible by protocols that "reserve" bandwidth for the application's needs, at a cost. In the

meanwhile the "free" protocols will still be used for applications that do not need instant or real-time results, like email, shopping, or simply browsing (although all these can benefit as well from RSVP, for a price of course, as a matter of fact some Web Sites will very likely start providing "RSVP-enhanced shopping experiences" or "RSVP-enhance video-conferencing customer service" in order to attract more clients). Also, the revenues generated by RSVP will allow for more investment in technologies that will improve the Internet at an even faster rate than now.

Application Providers will spring up to provide for the needs of not only software that can be downloaded and then run, but most prominently for network-aware applications, applications which download only the parts of themselves that are needed for a lightweight and basic get-the-job-done solution, and which can later download from the network parts as needed (like a spell-checker or a formatting extension or data type conversion module). Many of today's traditional software companies will have to adapt to this new scheme. Also, how about Pay Per Use? (PPU)

Systems on a chip will become a big market boom, but in order to become a no-questions-asked option for designers it'll need a similar type of inter-component interoperability as what already exists in the software world. The goal here is to take the same approach of building PC systems out of standard off-the-shelve components to the chip level, where designers can pick and mix what they believe is the best from many "module" offerings (there is a connection here between Object Technology, Component Technology, and my Abstract Units and Tunnels technology). Note that there is already an effort under way called the Virtual Socket Interface (VSI) to provide exactly this level of inter-operation at the system-on-a-chip level. Once you couple this with technology that can pack the equivalent of many orders of transistors more into smaller cell sizes, we'll truly end up with whole computing devices that we can wear in our clothing and that are always able to connect to "The Network" wirelessly.

HTML will eventually be passed out and replaced by executable content in the form of Applets (which today's best way to do it is using Java due to its cross-platform executables, as opposed to all the different versions of ActiveX which require ActiveX executables for each architecture). The trend here is from static pages to dynamic content (a good reason why JavaScript (which has nothing to do with Java) has picked up some steam in providing basic interactivity for some web pages). HTML will eventually only be used for simple static pages and as placeholders for Applets. The future Web will essentially be a world-scale application made out of millions of smaller ones sitting on web pages, all communicating with each other very much like the neurons in our brains).

"Pull" and "Push" methodologies will disappear (actually, they will be buried behind what the user sees, they will become "invisible" to the user) behind "Virtual Displays" (VDs) that will be delivered to your "display" to perform a specific function.

These VDs will take various shapes, and their enabling technology will be a standardized component model (like today's JavaBeans), an inherent (and expected) Internet connectivity, and standard ways to communicate with each other (like TCP/IP, IIOP, HTTP, etc…). These VDs will reverse the way we view information today on the web. Today we use a browser to get to the data, with VDs it's the data that comes and shows itself via its own "browser" (or let's call it "Virtual Navigator" (VN). A side effect of these will be a tremendous amount of incredible new applications unlike anything anyone can imagine today, and the web will truly become a dynamic medium for data access and manipulation (note that today's browsers will still exist, but only for those applications where still makes sense to use them for).

Java VM → JIT (and derivatives) → HotSpot → Java Chips → JavaBlaster → Java Converter

---

We need to identify incentives to require every new building built to have direct Internet connection capabilities (either through cable, digital phone lines, or satellites). And have each room contain at least one fiber optic connector (or maybe just 10 Base-T at minimum?) for appliances (PCs, NCs, PNSs, DVDs, etc…). Some of these incentives could come from the fact that technical people specially in the computer consulting fields will very likely be willing to pay more money for an apartment if it has a direct connection to the Internet whose usage is already included as part of the rent.

Abstract Units and Tunnels introduce the topics of having inherent built-in ways to deal with copyright information, licensing, distribution, payment, etc…

Need to talk about the business side of things on every tech section, to show that there is a reason for everything. i.e.: we don't do things so that they're "cool", but rather because they serve a purpose.

Will we standardize on a "Computing Block" you just buy off the shelf and use, when it breaks you throw away and the application keeps working normally like if it was a RAID disk???