The performativity of code: software and cultures of circulation

Adrian Mackenzie

Introduction

In a recent novel, *Distraction,* Bruce Sterling describes a future scenario in which information networks have turned bad (Sterling, 1998). By 2044, China has flooded the world's computer networks with pirated copies of commercial software. The US economy, long supported by monopolistic intellectual property arrangements, has consequently collapsed. A populous underclass of unemployed technicians, programmers and engineers calling themselves Moderators roam through splintered urban and rural zones in bands harvesting and recycling technological junk and waste products, extracting energy, discarded components and materials and converting them into tools, energy and materials for their own use. At the centre of post-consumer nomadic Moderator life stands an important infrastructural component: the servers. The Moderators' servers monitor, collate and record the status of members of the community. Individual status fluctuates in real-time in response to continual polling by the servers of the community's opinion of individual contributions to the life of the collective.

Sterling's scenario is not too remote from some contemporary realities. Although software developers do not in any large measure regard themselves as a disenfranchised nomadic underclass, and property rights have not collapsed (in most domains), software cultures increasingly display the kinds of complex collective orderings that Sterling describes. Important sectors of software production are in transnational migration – between UK/Europe/US and Asia – and in trans-institutional movement between corporations, universities and loose aggregates of paid and unpaid workers. Software workers move from Asia to Europe and the USA under special immigration quotas at the same time as European and American companies outsource their software development

to software houses in India or the Caribbean. At the same time, self-organised software projects relying on network servers generate large-scale code objects such as operating systems and web servers which leading computing enterprises such as IBM and Apple take up, sponsor and promote.[i] Programmers such as Linus Torvalds, Larry Wall, Alan Cox, and Richard Stallman reach celebrity status within programming cultures, and minor stardom in the mass media.

Although there has been a profuse acknowledgment of the mobility, dynamism and operationality associated with information networks (Poster 2001; Castells, 2001; Lash 2002), understanding the cultural specificity of software or code objects remains difficult. That specificity is problematic: software has been heavily commodified since the early 1980s when personal computers first began to change from hobbyist devices to office computers running spreadsheets such as *Visicalc* and word processing programs such *WordStar* (Lohr, 2002). Yet software resists commodification in various ways – the tools needed to develop software are widely available and proprietary claims over algorithms are hard to defend. Software often remains invisible because it is infrastructural (Bowker & Star, 2000) and distributed through many different channels. Software production and consumption is also the object of intensely embodied identifications and personal styles(Perl programmers vs Python programmers, Java programmers vs C++ coding). It is divided into platform-specific subcultures (Unix programmers vs Windows programmers). It is at once thoroughly pervaded by interlinked global standards and conventions (such as communication protocols), and anarchically polymorphic and mutable. New conventions constantly compete with existing standards. Numerous debates, identities, forms of commodification, capitalisation, and regulation swirl around software. Establishing a critical yet synoptic cultural viewpoint on software and code as operational objects remains difficult.

This paper investigate some contemporary migrations, translocations and twists in the technoscape[ii] associated with one highly complex and polymorphous code object, the Linux kernel, the core component of the GNU/Linux operating system. Linux is often cited as the primary example of free software or 'Open Source Software' (itself a term coined by commercial software producers and computer book publishers in 1998 to dispel the more financially disturbing connotations of 'free' software during the dot-com boom). Linux has made the cover of *Time*, it has frequently been discussed in editorials,

received the Golden Nica prize of Ars Electronica (Ars Electronica 1999), it has generated substantial speculative activity on financial markets during the late 1990s (Taylor, 1999). It has enrolled tens of thousands of programmers and software developers in (mostly) unpaid software development projects. At the same time, Linux, and the other major open source success story, Apache, a web server, have been fairly quickly slotted into corporate software production at companies such as IBM, Compaq, Hewlett Packard, Apple, and Sun Microsystems. In the meantime, open source software has generated huge quantities of meta-commentary about software (of which this paper forms a part), and challenged the blackboxing of commercial software in important domains.

Technically, the Linux kernel, dating from late 1991, is not an original or totally new thing. As is well-known, it explicitly *clones* another older mainstream operating system, Unix, itself dating from the late 1960s. Organisationally too, Linux is a highly centralised project in many ways. The very term *kernel,* or core, implies a degree of centring that more radical technical architectures have dispensed with. Even after 10 years, the project is closely controlled by and figured in terms of one person, the Finnish programmer Linus Torvalds. Neither the minor celebrity of Torvalds, nor the relatively conventional architecture or technical features of Linux itself constitute radical innovations. So how does something such as Linux become the object of intense feeling? How does it manage to enlist hackers and programmers to work inordinately hard on it, and at the same time become something that Wall Street, West Coast venture capital, the Pentagon, the European Union, schools in Goa, the Japanese government, anti-globalisation protesters in Genoa and many other institutions (but not US Congress (NewsForge, 2002)), organisations, groups and individuals see as a desirable, as a solution to their problems?

This paper argues that the ongoing development of Linux can be understood as a partial solution to a more general problem concerning the relation between information infrastructures and conventions, proprietary objects, and the gendered identity of 'knowledge workers' such as programmers. Linux represents a form of collective agency in the process of constituting itself performatively. By virtue of its operationality, I will suggest that in semiotic terms Linux amounts to an *indexical icon,* something that recursively refers to a description of itself. Put more concisely, it is 'a self-reflexive use

of reference that in creating a representation of an ongoing act, also enacts it' (Lee & Lipuma, 2002, 195). The performative constitution of collective agency associated with Linux is complex. Just as the Linux kernel works by co-ordinating and scheduling computing processes of many different kinds, so too the ongoing development of Linux involves co-ordination and scheduling of many different programmers' work. Together the social organisation of code work and the operating system itself constitute a performative process in which describing and enacting what is described coalesce. Importantly, performativity works by covering over and holding something in place. In this regard, as we will see, the capacity of the Linux collectives to challenge gender norms remains severely limited.

The problem of operationality

Information networks take many different forms and rely on a variety of different materials to propogate and circulate messages (ethernet, cable, modem, wireless, satellite, radio, microwave, optical fibre,etc). These different materials generally connect into a relatively small range of computing hardware (Intel, Sparc, PowerPC) and operating system platforms (Unix, Windows, Linux, MacOS). As Lash says, platforms constitute zones on which 'technological forms of life' depend (Lash, 2002, 24).[iii] Linux constitutes one such platform.[iv] It is an operating system currently operating at tens of millions of different points in the information networks. It constitutes a domain, I will argue, where problems of property, commodity specificity, communication and production are played out fairly intensely. Like other operating systems such as those produced by Microsoft or Apple, Linux puts layers, many layers in fact, of code between the loosely assembled commodity hardware (cpus, motherboards, disk drives, network interfaces, various parts, graphics and sounds cards, etc) and the application software (various programs and applications such as webservers, databases, word processing programs) on which user-interfaces are focused. Linux differs from operating systems sold by Microsoft and Apple in that it runs on a range of different hardware platforms, ranging from handheld computers (Shah, 2002), through game consoles such as Sega Dreamcast and Sony PlayStation to IBM supercomputer clusters. Lately Linux 'ports' (or adaptations) have appeared for various embedded or realtime platforms used in controlled non-computing devices or systems.

Apart from the incessant struggles over market share between Microsoft and Apple, and the endless debates and campaigns that struggle over the relative merits of Mac vs. Windows, the question of what operating system runs on a given hardware platform would hardly appear to be a pressing cultural question. Linux emerges from the relatively affluent domains of university computer science departments, corporate IT departments and research labs, and various software US computer companies. Linux has however, to a certain extent, gone beyond these privileged domains of technical production. One question is how that has happened. Lash suggests that '[i]n the representational culture the subject is in a different world than things. In the technological culture the subject is in the world with things' (Lash, 2002, 156). Within technological cultures, operating systems and server software constitute just such things or cultural object. As culture becomes 'operational', or as information technologies become more cultural, that is, as they merge into wider circulatory practices of ordering and coding, of representing and regulating differences in some ways and not others, erstwhile infrastructural things like operating systems, protocols, algorithms and code figure more significantly.

In what sense does an operating system (or for that matter, a genome, a public database, a stem cell, etc) constitute a cultural object? An operating system is hardly a medium, in the sense that television or newspapers are media . Nor is an operating system a message whose content or meaning can be analysed. Certainly, operating systems are often products that are packaged, circulated, regulated (as in the US Attorney-General's anti-trust action against Microsoft Corporation's Windows operating system) and consumed on a massive scale. Yet at the same time, as Linux indicates, an operating system may not be reducible to a conventional commodified object if it constantly modulates as it moves through a distributed collective of programmers and system administrators. By contrast, the same thing cannot be said for computer hardware. Almost without exception, computer hardware is commodified and its production is industrial.

An operating system such as Linux challenges the typical analytical separation between production, circulation and reception (consumption, use, spectating or audiencing) in a number of ways. If much social and cultural theory relies on that separation *de facto,* the question remains: what kind of culture object is Linux? Through what modalities

(technological, compositional, social) and at what sites (production, reception, as an object in its own right) should it be analysed?

Software performance and performativity

As a technical object, the principal claims made for Linux have been remarkably consistent. The claims are that Linux is 'free' and that it performs better than similar commercial products such as those made by Microsoft or Sun. Leaving aside for moment the question of whether or how Linux is 'free', Linux has been and continues to be clearly figured in terms of its technical performance. For instance, a recent advertising campaign for the database and 'enterprise infrastructure' company Oracle Corporation claims that Linux and Oracle together are 'unbreakable' (Oracle Corporation, 2002). Computer companies ranging from IBM, Compaq, Hewlett Packard, to Dell, as well as retailers such as Wal-Mart all currently sell Linux on this basis. For the most part, they market Linux to corporate and government clients. However, in some places, consumer sales of Linux have become significant. In China, where legal controls over intellectual property are beginning to be enforced as a condition of China's entry into the World Trade Organisation, a distribution Linux called TurboLinux outsells Microsoft Windows (TurboLinux, 2002). We could say then that as a cultural object, Linux figures largely in terms of performance combined with low cost. How can that performance, denoted by terms such as 'unbreakable', 'proven performer' (Redhat, 2002), or 'stability,' be analysed without falling back on a naturalised or technologistic understanding of performance?

Broadly speaking, the technical performance of Linux is represented within a discursive formation associated with information and communication processes. That formation discursively situates and complicates any unmediated technical 'performance' of Linux. Crucially, I am proposing that technical performance itself is coupled, in ways that need to be analysed, to *performativity*. As power becomes more 'performative' ('power itself is no longer primarily pedagogical or narrative but instead itself performative' (Lash 2002, 25)), information and communication systems and networks come to be one important venue in which the performativity of power is enacted. What does it mean to say that an object becomes performative? Given the many ways, sites and levels at which performativity works, and the large-scale differences of class, race, gender and sexuality for which the conceptual applications of performativity have been developed,

how is something quite infrastructural like Linux at all significant in the contemporary performativity of power?

One reason to speak of performativity is to account for the effects of power in terms of objectifications of linguistic praxis. Across the wide ranging discussions in social, political and cultural theory of performativity over the last decade or so (Butler, 1997; Derrida 1972), this dimension of performativity has been continually emphasised. As a performative object, Linux would have to be understood not just in terms of the meanings ascribed to it, or in terms of the effects of its performance on the movements of data and information in communication networks. Rather it would be an objectification of a linguistic praxis, as a 'self-reflexive use of reference that enacts the act that it represents.' Hence analysing Linux's performativity would involve extending a speech-based notion of performativity to the discursively mediated practices of coding or programming, distributing, configuring and running an operating system. As Lee and Lipuma write:

> The analytical problem is how to extend what has been a speech act-based notion of performativity to other discursively mediated practices, including ritual, economic practices, and even reading. What is interesting about performatives is that they go beyond reference and description-indeed, they seem to create the very speech act they refer to. (193)

What would be the 'act' of something like Linux? The insistent claims about superior technical performance and being 'free' mentioned above constitute, I would suggest, the public face of the 'speech act' in question here. Insofar as being-free and being technically better matter, Linux succeeds as a speech act. However, the performativity of Linux is complex. Arguably it is not constituted principally through meaning-making processes such as narrative, but rather through what Lee and LiPum term 'circulation'. They write:

> Performativity has been considered a quintessentially cultural phenomenon that is tied to the creation of meaning, whereas circulation and exchange have been seen as processes that transmit meanings, rather than as constitutive acts in themselves. Overcoming this bifurcation will involve rethinking circulation as a cultural

> phenomenon, as what we call cultures of circulation. (Lee & LiPuma, 2002, 192)

Lee and LiPuma argue that circulation produces performative effects. That is, processes of circulation themselves objectify linguistic praxis. They enact something. If we accept that information and communication constitute a central venue for the performativity of some important contemporary forms of power, then the circulation and exchange of software and code involved in the infrastructure of communication could well be analysed there in performative terms. From this perspective, the explicit claims about Linux's technical performance, as they appear in advertisements, editorials, newsgroups, how-to manuals and popular press accounts, would be only a secondary effect of the more primary, collectively performativity of practices channelled through computer code.

Circulating, distributing, and co-ordinating code

The central claim that I am advancing here concerns the set of practices that produce, circulate and consume Linux. As an operational object serving as a platform, Linux quite literally coordinates the circulation of specific social actions pertaining to information and communication networks. At the same time, coordinated actions centred on Linux constantly modulate it as an object in self-referential ways. For instance, development work done by programmers on the Linux kernel modifies the very platform on which they do their programming. The code they work on can be seen as an intricate description of the many features of a highly configurable technological ensemble. That features of that description constantly change through 'kernel hacking.' Such changes in turn affect how the ensemble itself operates. Because it operates differently, Linux as an operational act changes. Each *release* of the kernel (there have been dozens of releases) circulates differently because its features have changed. New hardware configurations, new communication protocols and new kinds of connectivity are constantly incorporated into the kernel. Through that modulatory circulation an objectification of linguistic praxis occurs which gives rise to both the effects of technical performance and the freeing up of the proprietary status of the code.[v]

The agencing effect of performatives rests on their capacity to 'objectify' some kind of praxis, typically linguistic practices. This objectification, however, only succeeds

provisionally or partially. Judith Butler writes:

> If a performative provisionally succeeds (and I will suggest that "success" is always and only provisional), then it is not because an intention successfully governs the action of speech, but only because that action echoes prior actions, and *accumulates the force of authority through the repetition or citation of a prior and authoritative set of practices*. It is not simply that the speech act takes place within a practice, but that the act is itself a ritualized practice. What this means, then, is that a performative "works" to the extent that it *draws on and covers over* the constitutive conventions by which it is mobilized. (Butler, 1997, 51)

Could the analysis of performatives extend to the repetition or citation of a prior authorizing set of practices in code? How would the 'repetition or citation' involved in Linux help us understand its success? What is cited or repeated? What is covered over? What extra purchase would the idea of the partial workability of performatives give us in understanding collective investment in Linux?

'Distros': repeating and citing Linux

One curious feature of Linux is its propensity to circulate in many different forms. At one end of the contemporary spectrum, we could point to recent distributions of Linux as an artwork. An online and radio broadcast performance of Linux ran during for the first half of 2002 called RadioFreeLinux (Radioqualia, 2002). The work consisted solely of the source code (the program as written and read by programers) of the Linux kernel (the central part of the operating system that interfaces between the endless variations in hardware and the application programs and services that users run on their computers) being read out line by line over various radio and streaming web-radio sites.

Although not particularly fascinating to listen to, the work signals that the value of computer code ('source code') is changing. On the one hand, it is 'free' in the sense of freely available to be copied and distributed. It is often given away on the CD-ROMS stuck to the front of popular computer magazines. On the other hand, source code possesses to a greater or less degree some social or cultural value apart from economic value. The title *RadioFreeLinux* suggests that source code should circulate or be

broadcast like news or entertainment. The work is not an isolated aberration. In 1999, the Linus Torvalds was awarded a significant new media art prize at Ars Electronica. Linux had almost become, perhaps not quite, an object of aesthetic value:

> The Jury of the .net category awards the 1999 Golden Nica to Linus Torvalds as representing all of those, who have worked on this project [Linux] in past years and will be participating in it in the future. ... It is also intended to spark a discussion about whether a source code itself can be an artwork. (Ars Electronica, 1999)

Even if art prizes are partly driven by the desire for frequent revolutions and innovations, what does it signify when source code becomes an artwork? Given that source code for programs has been around for at least 50 years, why has source code become something which people want to 'read' now?

At the other end of the spectrum, early in 2002 Sony Corporation announced that it was releasing a version of Linux for the games console, PlayStation2. Typically gaming development environments such as *ProDG* for the PlayStation cost between $US5000-10000 for restricted developer licences (SN Systems, 2002). Sony itself licences games development in order to control the quality of the games commercially released for the console. In a shift of licencing policy,the Sony announced Linux for PlayStation by saying:

> The PlayStation 2-specific libraries will be released under the LGPL; there are no proprietary licenses involved. Sony's distribution of Linux is based on Kondara, which in turn is based on Redhat. The documentation with this kit will give all the same information about the PS2 hardware that Sony provides its licensed game developers (but it won't give access to the system's anti-piracy mechanisms). This will include full details on the PS2's proprietary Emotion Engine core instruction set, the Graphic Synthesizer, and the Vector Processing Units. (Wen, 2002)

The migration of open source code onto the proprietary hardware PS/2 platform is not unique. As mentioned above, Linux has been 'ported' to many different proprietary hardware platforms ranging from PDAs to mainframes.[vi] This announcement points to something else: 'Sony's *distribution* of Linux is based on Kondara, which in turn is based on Redhat'. This avers to the fact that Linux exists in many different 'distros.' Kondara and Redhat are existing distributions or repackagings of the Linux kernel along with

associated software. Thus the PS/2 distro figures as just one amongst around at least 100 different commercial and non-commercial distributions of Linux (Distro, 2002). Some of these, such as PS/2 Linux, have the specific purpose of making a hardware platform more widely programmable and less of a blackbox. Some concentrate on particular application domains such as 'desktop users'. The European Commission-funded *Agnula* distribution focuses on audiovisual computing. Others have specifically a national or language focus. TurboLinux for instance, as the first Linux distribution to provide Chinese language support, focuses on selling Linux in China (Turbolinux, 2002). For the last 5 years, new distributions of Linux have appeared and disappeared at a high rate. Most people work with Linux by downloading a pre-packaged branded distribution such as Redhat, Mandrake or Debian, and then configuring the system to their own needs. It is also possible to build the operating system by downloading the kernel source code (www.kernel.org) and some other pieces of software independently of any of the branded distributions.

How can we understand the existence of this multiplication of different incarnations of the same thing,the Linux kernel, ranging from an audio broadcast, to quasi-proprietary Sony distribution, then to dozens of branded distributions, and finally to the source code itself, readily available from many different websites, ftp servers and mirror websites? Linux circulates as artwork, as commercially packaged commodity sold by many different companies, and as freely available source code files, constantly worked on and exchanged using sophisticated software mechanisms of co-ordination, scheduling and organisation running on Linux servers. The commercial distributions have largely come into existence as a way of capturing the 'free labor' embodied in Linux. Without violating the fairly stringent GPL licences which prevent the software itself being sold, companies like Redhat and IBM effectively give away Linux distributions, but charge for the support services needed to configure and maintain them in operation (Taylor, 1999). At the same time, the free work on the various components of Linux continues, and the changes constantly made to it by numerous programmers are successively incorporated into new releases of the commercial and non-commercial distributions. The economics of this process were subject to strong stock market speculation during the late 1990s (Taylor, 1999; Redhat, 1999), and extensively analysed by academic researchers.(Tuomi, 2000).

The internet-based co-ordination mechanisms that permit collective development work

on the kernel have been fairly thoroughly described in both semi-popular and academic accounts of open source (Moody, 2001; Bezroukov, 1999). For the moment, the important point is that the circulation of Linux in dozens of different distributions connects technical performance and the performativity of Linux as a cultural object. The distributions configure Linux so that it can circulate across different hardware platforms, and between different cultural, institutional and national domains. Each new distribution, and each successive release of an existing distribution incorporates and repeats the conventions embodied in the Linux kernel but adapts those conventions to a slightly different situation – a different hardware platform, a different language grouping, a different kind of computing task (supercomputing, office productivity, eCommerce, genomic sequence searching and alignment).

Linux: an object out of control?

Accounts of science and technology over the last decade such as (Latour, 1997), (Haraway, 1997) and (Lash 2002) have sought to analyse why certain contemporary objects like Linux are 'out of control'. By 'out of control', they mean that these objects generate unexpected consequences. They proliferate at a rate that cannot be accounted for by theories of innovation based on diffusion. Such accounts argue that technical objects like Linux become unstable and proliferate as a consequence of the translation or objectification of social and cultural relations into and through them. As Lash puts it, objects get out of control as an unintended consequences of modern rule-regulated reflexivity:

> [R]eflexivity involves the reflexive monitoring of the object by the subject, in which the subject subsumes the object under rules. ... The more we monitor the object, the more the object escapes our grasp. ... This moment of contingency, is where the object, or the self, escapes the cognitive categories of the subject, is indeed aesthetic. (Lash, 2002, 50)

As an object, the Linux kernel incorporates a tangled web of rules, conventions, standards, and protocols pertaining to information and communication networks, as well as specific features relating to the commodity computing hardware. The conventions range from the fundamental binary abstraction on which all code operations rest through

to the highly conventional protocols for transactions and messages on the information networks. These conventions are defined by regulatory and standards organisations such as the ISO (International Standards Organisation), the W3C (World Wide Web Consortium), IEEE (Institute for Electrical and Electronic Engineering), and ANSI (American National Standards Institute). The conventions and protocols attempt to subsume information systems under rules that render those systems inspectable or reportable. So, for instance, Linux 'aims for POSIX (Portable Operating System Interfaces) compliance' (Linux Kernel, 2.4.x, 2002).

Do the conventions present in Linux explain its proliferation, its propensity to 'escape our grasp'? If, as Lash writes, 'the more we monitor the object, the more the object escapes our grasp', it could also be added that the object's escape from 'the cognitive faculties of the subject' is not an accidental loss of control, a result of risk and uncertainty. Rather, it flows from the performativity of circulation. If the notion of performativity has any traction in explaining 'cultures of circulation' such as those associated with Linux, it might be that it provides a different way of apprehending the proliferation and complexification of objects. My analysis of Linux as a collective formation in the process of performatively constituting itself hinges on this point. The kernel institutes a performative act to the extent that in describing a nexus between commodity hardware and conventional, rule-governed orderings and movements of information, it also enacts that nexus. At core, the line between code object – the kernel – and code subject- the hackers and programmers personified in the figure of Linus Torvalds – wavers uncertainly. To be a hacker working on Linux means in some way to see Linux as a more than just an object to be apprehended cognitively, but as a form of life which involves challenging norms of property ownership and corporate organisation of work (Moody, 2001; Himanen, 2001).

What Linux covers over

Performativity undermines the prerogative of either a subject or an object to give meaning to things. The agential effect of performativity arises first of all, as we have seen, through repetition and citation. Linux repeats itself across platforms, and in different contexts. But another dimension of its performativity needs to be analysed. Performatives also 'succeed' not only by citing, or enacting through describing, but by

'covering over,' as Butler suggests, the 'authoritative set of practices' which lend force to the enacting. What 'authoritative set of practices' or 'constitutive conventions' does Linux cover over?

Judging by what is said in advertisements, in newsgroups or in editorials and essays about Linux, there would be some justification in thinking that the history of Linux, its biography as a technical object and commodity, is brief, linear, uncomplicated and dates from around November 1991. Around this time, the Internet first begins to enter into public awareness. The history of Linux is often presented as a consequence of the co-ordination and circulation of code that the Internet permitted (Moody, 2001). However, as mentioned above, Linux clones an earlier computer operating system, Unix. Drawing on Unix, Linux sediments a complicated relation to proprietary hardware and software. But Unix was and remains not just a piece of software, but also an associated set of coding, software design and system administrative practices, sometimes referred to as the 'Unix philosophy'. These practices are highly gendered. Linux's 'authorizing context', to use Butler's term, is complicated because it combines regulatory practices and coding-hacking practices. As an event, Linux is complex in ways that the popular narratives of the 'cloning of Unix' (Himanen, 2001; Moody 2001; Lohr 2002) occlude.

The kernel archive sites, such as ftp.kernel.org, show that Linux has been through dozens of releases cycles, and hundreds of minor updates. What started as a file of a few hundred kilobytes in size in 1991 has grown to 22Mbytes by mid 2002. If we wanted to ask how Linux begins to take on its own voice, or how it begins to accumulate agential force, we might also look at both at what what that growth involves and covers over. The README textfile found in every release of the kernel source code since 1991 begins:

> WHAT IS LINUX? Linux is a Unix clone written from scratch by Linus Torvalds with assistance from a loosely-knit team of hackers across the Net. It aims towards POSIX compliance. (Kernel 2.4.28 2002)

First of all, although 'written from scratch', Linux heavily cites another operating system

or software platform, Unix. So all the distributions of Linux currently circulating, and competing with various degrees of commercial and non-commercial success are versions of systems loosely grouped under the name 'Unix', 'a Trademark of the Bell Telephone Laboratories' (Kernighan & Ritchie, 1978, ix). Unix itself dates from summer 1969 when two computer scientists (Dennis Ritchie and Ken Thompson) at the Bell Telephone Laboratories in New Jersey developed an interactive operating system.

Authorising context (a): interaction and economy

Why clone Unix rather than Windows, MacOS, or some other operating system? By the early 1990s, when the Linux kernel first appeared, Unix was widely used at universities and research institutions in USA and Europe. Why was it so widely used? Without going into the technical lineages of operating systems, and the constant tradeoffs and exchanges between academic computer science, computer manufacturers, the U.S. military and telecommunications companies, the first academic paper published by Ritchie and Thompson reporting on Unix in 1974 (Ritchie & Thompson, 1974) emphasised the combination of the power of 'interactive use' and economy in relation to hardware:

> Perhaps the most important achievement of UNIX is to demonstrate that a powerful operating system for *interactive* use need not be expensive either in equipment or in human effort: UNIX can be run on hardware costing as little as $40,000, and less than two man-years were spent on the main system software (Ritchie & Thompson, 1974, 365).

The 'interactive' character of UNIX was an 'expression of our desire for programming convenience' (Ritchie & Thompson, 1974, 374) rather than any predefined operational objectives for the system. In other words, the kind of work that programmers face was deeply embedded in the design of Unix in a number of different ways. Not only were the kinds of features and functionality available in Unix suited to programmers, Unix was itself coeval with a particular programming language, C, that became an industry standard and is used to write Linux code. In contrast to other commercial operating systems at the time, Ritchie and Thompson explicitly set out to design a system for 'programmers' convenience. Consequently, Unix took on a particular shape based on two major abstractions. These abstractions strongly affected Linux. The notion of everything

as a *file (*Kernighan & Ritchie, 1974, 2: TODO: check this), the notion of 'the process' (the other fundamental abstraction in Unix) and the provision of many tools and utilities for accomplishing specific program and system maintenance tasks still completely pervades Linux.

By the early 1990s, just before Linux emerged, Unix already had its own quite complicated history of exchange and value (see Salus, 1994). One of the standard undergraduate textbooks on operating systems at the time, Tanenbaum's *Operating Systems: Design and Implementation,* says:

> UNIX has been moved ('ported') to more computers than any other operating system in history, and its use is still rapidly increasing. (Tanenbaum, 1987,11)

Unix migrated to many different hardware platforms both because of its specific technical attributes of Unix and the Bell AT&T licensing arrangements that meant users needed to work with each other to keep their systems running. The 'interactivity' of Unix (its ability to schedule and carry out more than one computing task at once) and its 'portability' are attributes central to Linux. From the outset, they were coupled together in the way Unix circiulated. This combination of circumstances certainly forms a constitutive convention for Linux. The history of Unix lies at a complicated junction juncture between the organisation of capital, state regulation of communication, and the technical work of programming.

Unix' circulation between 1969 and 1983, was strongly affected by an anti-trust case initiated in 1949 by the U.S Federal Department of Justice against Western Electric and AT&T, companies that jointly owned Bell Telephone Laboratories. Between the first public release of Unix in 1971 and 1983, AT&T had been bound by the terms of a court judgment made in 1956 to distribute Unix at a nominal fee to anyone who wanted it. As a telephone company, they were legally prohibited from profiting from computer software. At the same time, because they were legally prohibited from making any profit on an operating system such as Unix, AT&T provided no support and no fixes for bugs in the system. Unix was unsupported. As Salus writes, '[t]he decision on the part of the AT&T lawyers to allow education institutions to receive Unix, but to deny support or bug fixes had an immediate effect: it forced users to share with one another. They

shared ideas, information, programs, bug fixes, and hardware fixes' (Salus, 1994, 65). In 1983, a second anti-trust action (similar to the one mounted currently running against Microsoft) divested AT&T of the Bell Operating Companies and changed the commercial status of Unix. In the aftermath of the second anti-trust action, the newly formed AT&T Bell Telephone Labs, no longer a legal a part of a telephone company, could licence Unix for any amount it wanted to. The charge for the all- important source code rose steeply into the order of several hundred thousand dollars and it prohibited the source code from being studied in university courses (Salus, 1994, 190) where Unix had become a standard teaching tool for computer science students.

I am suggesting that Unix is an important component of the authorising context which Linux emerges out of. Unix existed for a long time as something that the US government prohibited a telecommunications company from commercialising. AT&T Bell was compelled to give Unix away freely. At the same time, because AT &T could make no profit, it provided no support, so adopters of Unix were motivated to do their own maintenance and development. Consequently, Unix became a pedagogical object. It was used to teach computer science students at universities during the 1970s and 1980s about operating systems. Because it came with its own programming langugage, C, Unix was also a highly mutable, reconfigurable object. Computer scientists used Unix as a testbed to explore new protocols, data structures and user interfaces. Unix grews as a part of the emerging infrastructures of networked computer emerging in the late 1970s and growing during the 1980s into the internet. This aspect of the authorising context centres on a particular set of communication protocols: TCP/IP. According to Abbate's *Inventing the Internet,* they were first incorporated into Unix in the late 1970s, mainly through ARPA funding (Abbate, 1999, 133).[vii]

Generally, speaking the primary component of the authorizing context underpinning the performativity of Linux comes from the peculiar set of Unix practices which intimately linked producers and users/consumers of operating systems.

Authorizing context (b): co-ordinating and communicating

What does it mean to say that 'Linux is a Unix clone'? In cloning a computing environment, where does the line between what is cloned and what is not get drawn?

Linux according to Torvalds was a 'program for hackers by a hacker' (Torvalds, 1991), just as Unix was, according to Kernighan and Ritchie, a system by programmers for programmers' convenience. Given that Unix had, as Tanenbaum's text book states, already been 'ported' to more platforms than any other operating system 'in history', what was at stake in having it cloned by hackers?[viii]

A second crucial and interlocking component of the authorizing context which undergirds Linux's performativity resides here. Linux follows in the wake of other 'ports' of Unix to personal computing hardware such as Minix. But the first pieces of code that could recognisably be called 'Linux' were experiments with a particular feature of the Intel 80386 chip called 'task-switching'. In a 1991 interview, Torvalds said:

> "I was testing the task-switching capabilities [of the 80386], so what I did was I just made two processes and made them write to the screen and had a timer that switched tasks. One process wrote 'A', the other wrote 'B', so I saw 'AAAABBBB' and so." .... "Gods I was proud over that" (Moody, 2001, 36)

Although it is a clone of existing Unix systems which had become increasingly proprietary by the early 1980s, Linux took root in an individual desire to 'test the task-switching capabilities' of the i386 family of microprocessors produced by Intel. Why was 'task-switching' of interest? One of the fundamental Unix abstractions, 'the process', supports Unix interactivity. By using the task-switching capabilities of the Intel platform (which underlay and still underlies most personal computers), Torvalds could build from the ground up a concrete implementation of the fundamental process abstraction. Unix culture then could move out of the institutional and commercial environments in which it had been developed and regulated via this adaptation to a commodity consumer hardware platform. Via the appropriation of consumer computing hardware effected by Linux, the conventional object Unix begins to metamorphose into something more dynamic and processual. Linux, we could say, transforms the consumption of commodity computing hardware by opening a window onto the 'authorizing context' embodied in Unix. How is consumption understood here? In the case of Linux, consumption of commodity occurs through coding. Through code, the proprietary hardware specificities of different computational devices and peripherals are addressed.

The first release of the kernel in late November 1990 contains a warning about relations to the proprietary devices:

```
LINUX 0.11 is a freely distributable UNIX clone.  ... LINUX runs only
on 386/486 AT-bus machines; porting to non-Intel architectures is
likely to be difficult, as the kernel makes extensive use of 386
memory management and task primitives.
(Torvalds, 1991)
```

So, in the first publically released version of Linux, a key piece of code contained in the 'main.c' file sets the operating system time from a hardware clock:

```
/*
 * Yeah, yeah, it's ugly, but I cannot find how to do this correctly
 * and this seems to work. If anybody has more info on the real-time
 * clock I'd be interested. Most of this was trial and error, and some
 * bios-listing reading. Urghh.
 */

#define CMOS_READ(addr) ({ \
outb_p(0x80|addr,0x70); \
inb_p(0x71); \
})
```

Kernel 0.11, main.c

As the code comments say, this is 'ugly' material to work with (especially as a researcher!). The lines of code are executed when the operating system starts up. They actually refer to some other lines that relate to hardware specificity. If we trace the line 'outb_p(0x80|addr,0x70);' to their definition in another source file, we find them defined as:

```
#define outb(value,port) \
__asm__ ("outb %%al,%%dx"::"a" (value),"d" (port))


#define inb(port) ({ \
unsigned char _v; \
__asm__ volatile ("inb %%dx,%%al":"=a" (_v):"d" (port)); \
_v; \
})
```

kernel 0.11 /include/asm/io.h

The code becomes even uglier and less easily readable. The increasing unreadability shows something important. These instructions read and write to specific spatial locations on the Intel x86 family of CPUs (specifically, IO (input/output) ports). Some of

the very first lines of the Linux kernel are very closely tied to the deeply embedded specificities of the Intel 80386 chip. Frustrated by the opacity and obscurity of the hardware, Torvalds writes in the code comments, 'I cannot find out how to do this correctly.'

During early 1991, postings on newsgroups such as comp.os.minix, soliciting programmers to work on Linux, were framed in terms of the difficult specificities of hardware devices. This is Linus Torvalds inviting people to work with on what would later be called Linux:

> Do you pine for the nice days of minix-1.1, when men were men and wrote their own device drivers? Are you without a nice project and just dying to cut your teeth on a OS you can try to modify for your needs? Are you finding it frustrating when everything works on minix? No more all-nighters to get a nifty program working? Then this post might be just for you :-) (Torvalds, 1991)

This first announcement and call for volunteers - 'men' - focuses on the all-night pleasures of low-level system programming of 'device drivers', the parts of an operating system that speak to specific pieces of hardware rather than higher level abstractions. The mention of a time when 'men ... wrote their own device drivers' presumes, first of all, a time when men had access to the device in question. In the ten years since, as the source code for the kernel has grown from a few hundred kilobytes (220kB) to approximately 20MB in size, code that addresses proprietary device specificities have been added to the system. As Linux is 'ported' onto different platforms, new hardware specificities have to be addressed. At the same time, assumptions about hardware specificities built into previous versions of the code have to be rendered explicit and then moved out of shared code into specialized areas.

A final facet of the constitutive conventions at work in the performativity of Linux must be mentioned. The Linux kernel is deeply tied to a gendered corporeal set of practices of programming work. The mention of 'all-nighters', of a time when 'men wrote their own device drivers', reminds us that Linux is a program by men for men who like to play with computing hardware. The specificity of the commodity computer hardware for which Linux hackers write device drivers correlates directly with the desire to 'modify

an OS for your [own] needs'.

Conclusions

I started out from the question of how computer code could be a cultural object. The background to this question comes from more general accounts of the operationality of 'the information order' (Lash, 2002, 4) or network societies (Castells, 2002). In certain ways, the Linux kernel reveals symptomatic features of the contemporary organisation of material culture. In production, circulation and consumption of information, in the ways that things like Linux mediate informatic sociality, and in the ways they articulate new intersections between the different sets of practices and conventions, code objects demonstrate that operationality depends on the constitution of collective agency. While the technical performance of Linux versus other operating systems is a matter of endless debate, its performativity, its capacity to enact what is represents or describes, is harder to contest. The proliferation of distros, sites, enterprises, products and designs associated with Linux attests to that performative force.

The main thread of my analysis of Linux locates operationality within a specific culture of circulation which works performatively. Extending performativity from spoken or written texts to objects is not new. Latour, for instance, writes: '[p]rograms are written, chips are engraved like etchings or photographed like plans. Yet they do what they say? Yes, of course, for all of them-texts and things-act. They are programs of action whose scriptor may delegate their realization to electrons, or signs, or habits, or neurons' (Latour, 1996, 223). In claiming that 'programs' do what they say, Latour attributes performativity to objects insofar as they 'realize' specific social actions. The gloss I would add to that point is that realizing programs of action is complicated.

Augmenting the claim that objects act and speak by virtue of delegation, I would suggest that the performativity of a cultural object such as Linux can be analysed as a *partial* solution to a problem of social identity for information elites. Caught between highly commodified hardware production, the stringent regulation of intellectual property, and an ethos of free-wheeling creativity and autonomous work, hackers have stabilised some things – gender norms – and radically reorganised others – the production and consumption of software. If performativity involves an objectification of social praxis,

then the stability or durability of that objectification depends on the citation and repetition of 'a prior authoritative set of practices', to use Butler's terms. Linux could be seen as standing at the point of convergence of practices coming from academic computer science (embodied in Unix) and large scale production, circulation and consumption of computers as consumer electronics. Linux constitutes a partial solution to the problem of how different authorizing contexts coming from the quasi-academic ethos of Unix computing and its coding conventions, proprietary licencing of software, the plethora of commodity computing hardware and protocols for networking can be articulated together. Linux remains a partial solution since the programming collective has been unable to overcome its own limitations in regard to gender. The most egregious feature of Linux – the fact that the credits listing is almost with exception male – shows that the objectification of social practice synthesises some things with great novelty but keeps others static.

As a postscript to the opening scenario from Bruce Sterling's *Distraction*, as mentioned above, a condition of China's entry into the World Trade Organisation is a strengthened regulation of intellectual property law. The growth of Linux in China, and its recent adoption by the Being city government suggests that Sterling's fiction might miss the mark on this point. However, the broader point of the scenario from *Distraction* probably does hold: a transformation in the organisation of material culture is involved in the emergence and dynamism of things like Linux. If 'a shift from a register of meaning to one of operationality' (Lash, 2002, 216) is underway, operationality will need to be understood not in the mathematical-technical terms that predominate in some accounts, but as an articulation of diverse realities.

References

Ars Electronica ' Golden Nica for Linux/ Linus Torvalds (Finland) "LINUX" http://prixars.aec.at/history/jurybegr/1999/E99www.html 12 Nov 2002

Benjamin Lee and Edward LiPuma, 'Cultures of Circulation: The Imaginations of Modernity' *Public Culture* 14(1), Winter 2002: 191-213

Bezroukov, Nikolai 1999('Open Source Software Development as a Special Type of Academic Research' *First Monday,* http://www.firstmonday.org/issues/issue4_10/bezroukov/index.html

---

iUnder the current circumstances, it becomes increasingly difficult to locate software or code objects as a part of monolithic globalizing juggernaut, flattening whole domains as it rolls across the face of the globe. Nation-states such as the U.S.A. attempt to break the monopolies of major software companies (e.g. the Department of Justice's anti-trust case against Microsoft), while self-organised groups of software developers clone whole operating systems in their 'free' time (for instance, the Linux operating system).

iiThe notion of 'technoscape' outlined by Arjun Appadurai in the framework of his influential notion of the 'ethnoscape' offers one possibility here: 'by technoscape, I mean the global configuration, also ever fluid, of technology and the fact that technology, both high and low, both mechanical and informational, now moves at high speeds across various kinds of previously impervious boundaries' (Appadurai, 1996, 34). If the well-known notion ethnoscape couples migratory movements of people to imagined movements ('going home', 'making a new life' etc), the technoscape couples, by analogy, movements of technology with imaginings of that movement.The notion of technoscape remains relatively undeveloped in (Appadurai, 1996). It refers only to a 'fluid global configuration' and the movement of technology across boundaries (between countries, between workplace and home, etc). The concept of technoscape needs to offers ways of talking about how technologies move about together with imagining how technologies move. To match the conceptual sophistication of Appadurai's understanding of the ethnoscape, the technoscape needs to countenance a topographically and materially diverse domain, marked by boundaries, thresholds and paths, populated by different imaginings, devices, infrastructures and systems in interaction.

iiiServers occupy the nodal points of information networks. Servers offer different 'services': that is, they configure different points of access, and channel different kinds of movement of data through information networks. Some servers send web pages, some receive and send email messages, some transport or 'stream' audio or video. Dozens of different protocols (ip, tcp, icmp, ftp, irc, telnet, ssh, ssl, rmi, rpc, smtp, nmtp, ...) regulate these movements of data.

ivA web associated with this article shows a table that classifies some typical materials that circulate around and through Linux. See: http://www.lancs.ac.uk/staff/mackenza/research/opensource_materials.html

vThe feedback loop between performance and performativity mentioned above plays an important role because it triggers constant modulations both in the object itself and the practices associated with that object. Linux gains increasing traction as a commodity partly on the basis of this performance, yet that performance relies on the performativity of the practices associated with Linux.

viSony is giving developers access to the 'proprietary Emotion Engine'. On the one hand, games platforms are centred on the production and manipulation of affect through visual, audio and tactile events. In game play, certain kinds of labour 'in the bodily mode' heavily interact with audiovisual (and tactile) information objects. Some of the most dense and complicated interfaces to information systems are found in gaming. On the other hand, as the fine detail on the PlayStation Linux site shows, Sony is not giving any programmatic access to the DVD or CD drives. Understandably, they are trying to prevent the PlayStation from becoming a DVD ripper, something that would undermine Sony's broader audiovisual entertainment business.

viiDuring the last two decades, these protocols have come to almost universally regulate the flow of data between hosts or servers on the Internet. These protocols provided a way of connecting many computers together, and ordering the movement of data (packets) between different parts of the network without making many assumptions about the content or internal structure of the data.

viiiExpressive traits of Unix enter Linux partly through university undergraduate courses on operating systems. Linux bears the marks of Linus Torvalds' close reading of Tanenbaum's academic textbook on operating systems during 1990. In the mid 1980s, Tanenbaum, a computer science academic in Amsterdam wrote a Unix clone called Minix as a companion to his textbook and in order to be able to teach students about Unix as an operating system without incurring AT & T's newly imposed licensing fees. The source code of AT&T's Unix was not available to students. By contrast, the source code for Minix was publically available for a small fee so that students could understand how the system

worked. Thus by 1990, as an undergraduate computer science student at Helsinki University, Linus Torvalds himself had access to a pre-existing clone that could run on an Intel-based PC, and a textbook which gave a complete account of the architecture of a Unix operating system to explain it. But why then did Linux come into existence, given that Minix already ran on personal computers and provided Unix-style computing to almost anyone, students included, for a small fee? Tanenbaum's Minix (which had tens of thousands of users) was a system designed to support teaching about operating system design. It downplayed hardware specificities so that the system would stay simple and easy to understand. It was only slowly enhanced to deal with a wider range of hardware devices. If Minix emerged in response to to the licencing restrictions on Unix source code, Linux was defined in relation to the proprietary nature of hardware. Linux was a system that transformed a didactic project with restricted circulation (Minix) into a widely circulating and rapidly modulating object.