The Mechanics of Order

The Mechanics Of Order

An inquiry into the utopian possibilities of the free and open source ecology

TORRANCE HODGSON

Victoria University of Wellington

Masters Thesis, May 2010

Contents

Introduction 1

I. The Exploration 11

In which we meet the main protagonists, glimpse upon the realm, and embark upon our story

II. The Abstract Machine 37

On the problem of order, and its production in and through things

III. The Passport 57

Wherein virtual space becomes segmented and we discover the embrace of the user-space machine

IV. The Exodus 89

In which talk of escape renders the powerless as powerful, and the gatekeeper is transformed to maintainer

V. The Module 117

In which our adversary Complexity finds himself tamed by a happenstance of objects, borders and documents

Conclusion 147

Abstract

This is a study that concerns itself with two questions: how is order produced? and, is this order desirable? Contrary to many utopian methodologies that seek to elaborate 'what is not' but which 'ought to be,' this is a study that seeks to contribute to a *utopian mechanics* by way of studying extant subterranean practices or 'minor traditions,' by studying elements of 'what is' that may also form something of what 'ought to be.'

This study takes as its principal task to understand the production of order within a small free and open source project known as Compiz. It borrows from Michel Foucault, Gilles Deleuze and Félix Guattari to formulate the related concepts of the *machine* and the *abstract machine* in order to account for the ongoing production of order. These two concepts, following the lead of Bruno Latour, adhere to a 'flat social' ontology and bring forth the world of objects and space as being indispensible, alongside the members of Compiz, in accounting for the project's ordering.

The study poses three primary machines of order: the Passport, the Exodus and the Module. The Passport regulates access within the virtual spaces of Compiz and produces a role known as the 'gatekeeper,' one who may exercise a power both vicarious and precarious. The machine of the Exodus makes the threat of desertion a real and ongoing possibility and in this establishes an 'imaginary counter-power' within the group, undermining the power of the gatekeeper and recasting him as a steward of the code, as 'maintainer.' The third machine, known as the Module, is designed to minimise the complexity of the project by way of the spatialisation and organisation of the code, but subsequently effects a concomitant spatialisation and organisation of developers and projects, coming in the end to shape the large scale order amongst free and open source projects.

The study concludes by suggesting a 'present tense' and 'open ended' conception of utopia, in which both the machines of the Exodus and the Module — but not the Passport — would find themselves well placed.

Acknowledgements

I WISH TO FIRST THANK MUM for providing me with the topic of this thesis back in 2008, your support throughout and, most recently, for your thorough and very kind proofing of its final form. I also wish to thank Dad for our discussions that helped me clarify some of my earliest ideas.

To John, whose constant company helped me survive both my honours year as well as the fifteen months it has taken to complete this thesis, I wish to thank you. You provided both emotional support and, in the course of our many caffeine-induced conversations, helped form some of the most important theoretical aspects to this analysis.

To my friends, I wish to both thank you for your support and apologise for my absences and neglect as I struggled to see this project through to completion.

And last but certainly not least, I want to express my thanks to my supervisors, Chamsy El-Ojeili and Patricia Nickel, for your help, feedback and very generous enthusiasm as I worked my

way through each chapter. I most thoroughly enjoyed our regular meetings in which we discussed nigh on everything under the Sun and still managed to fit in serious discussion around my thesis.

Introduction

This is a study that concerns itself with two very simple questions: how is order produced? and, is this order desirable? It is a study that is, on the one hand, a sociology, a study of the production of social order amongst a loose network of programmers. It is, on the other, rooted within a utopian tradition, within an orientation that seeks social forms both more tolerable and more desirable than that with which we are confronted everyday.

These two traditions are often considered at odds, one given to the domain of dispassionate empiricism whilst the other relegated to the status of fantasy. Ruth Levitas, for example, has written of this perceived conflict between sociology and utopia:

> Sociology, surely, is a discipline of social *science*, and even those who doubt its scientific credentials, or question the meaning of scientificity itself would argue that it offers thick description and explanation of reality, of what *is*. Utopia, on the other hand, is essentially about what is *not*, and what *ought to be*.
>
> (Levitas, 2005; emphasis in original)

There is a different utopian tradition, however, that avoids these problems, one that is located at the interstices of two states: the study of *what is*, and the study of *what ought to be*. Such a project emerges in part from an orientation that asserts that no system is totalising, none achieves hegemony without the ongoing presence of an excess, a set of often fleeting practices, 'minor traditions,' that exist in *spite* of hegemonic practices. It is precisely in these minor traditions that this alternative utopian tradition finds its 'ought to be' that is also and already 'what is.' Within the cracks of the social lie glimpses of possible worlds, subterranean practices that, though only partial and though always and already mixed up with hegemonic forms, offer an empirical basis for an investigation of utopia. Stevphen Shukaitis writes,

> The task then becomes looking at the different existing forms of cooperative enterprise and social structures and asking how they might fit together into a general social vision or system […] [These include:] local community gardens, multitudes of cooperative and work collectives, the Mondragon, time stores and labor exchanges, […] the Kibbutzim, neighbourhood assembleas from Argentina, […] gift economies and exchange clubs, free stores, squats […]
>
> (Shukaitis, 2010: 307)

To this list we can add the topic of this study: the social ecology of free and open source software.

The origins of free and open source software can be traced to a rejection of the logic of Capital. In the early history of computers up until as late as the 1970s, the various corporations involved in the manufacture of computers relegated their software components to a secondary status. The software was indeed neces-

sary for their running but it was the hardware, they believed, that held the real commercial value. For this early period, the software enjoyed a status quite alien to the status of software today: it was often given away for free with computers, its source code very often accompanied its distribution, and the buyers of computers — who were mostly corporations themselves — often participated in the writing and maintenance of the code. During the 1970s, this view on software changed and it came to be perceived as valuable apart from the computer hardware itself. Copyrights came to be enforced, license agreements which restricted the manner in which the software could be used became a standard fixture, the source code of programs became a closely guarded secret, and buyers of this software no longer had the ability to alter its code. It was this progressive commodification of software throughout late 1970s that finally led MIT researcher Richard Stallman to initiate the GNU is Not Unix (GNU) project in 1983. Its aim was to construct an entirely free operating system modelled on the then-popular Unix system, rewriting its components bit by bit. While the GNU project was not wholly successful in its aims, it nonetheless laid the framework for the later development of the Linux operating system and a host of other projects, eventually leading to the formation of the free and open source ecology (Chopra & Dexter 2008: 12).

The term 'free and open source software' (FOSS) refers to a specific type of property relations concerning software. FOSS is required to be free in four distinct ways: one in possession of such software must be free to use it without restriction, free to study it, free to alter and improve it, and free to distribute it. FOSS software can still be bought and sold, but generally it is also free to acquire. The most important aspect to allowing for the four free-

doms is the presence of the software's 'source code,' which is the human-readable instructions of the program, and without which studying and altering the program would become for all practical purposes impossible. Software, therefore, is only considered free if it both allows for these four freedoms and if it also makes its source code publicly accessible or 'open sourced.'

Free and open source projects are characterised by some quite novel and utopian relations, which are the outcome of several baseline features. These baseline features include its property relations wherein the code and the means of production are fully socialised, the relative ease of finding virtual space in which to 'set up shop,' the ease with which most artefacts are duplicated, and finally the relative absence of coercion. Such features lend themselves to social relations which are radically decentralised, where participation is often open to anyone, and where work is voluntary, unalienated, and characterised by an amended communist ethos of 'from each according to their desire, to each according to their needs.'[1] It is according to these sorts of liberatory and anarchistic relations that the immensely complex engineering task of producing the Linux operating system continues to proceed to this day, involving the efforts of many tens of thousands of programmers, hundreds of projects, and producing an artefact whose production by capitalist means would have cost an estimated US$10.8 billion (Hale-Evans et. al., 2008).

1. I must immediately note an important disclaimer. More and more FOSS projects are attracting the interest of Capital, wherein companies are subsequently employing their own staff to contribute to these projects so as to tailor the programs according to their needs. For these programmers, their participation is no longer voluntary and their coding efforts are directed towards the interests of their respective companies, which include the likes of IBM, Red Hat, Sun, Oracle, Novell and Intel (Corbet et. al., 2009).

Of these thousands of projects, this is a study of but a single, small project. Its name is 'Compiz' and it was initiated by programmer David Reveman who opened up the project for community participation in early 2006. The program was a type of 'compositing window manager' which brought three dimensional capabilities to desktop windows, allowing for things like window transparency, impressions of depth and a variety of useful desktop effects. The project quickly gained considerable attention and attracted a number of developers who wished to volunteer their time. The project serves as a fantastic lens into the everyday practices of free and open source developers and the means through which their projects come to be ordered. But, moreover, Compiz is a rather dramatic example, as it underwent a split or 'fork' — a relatively rare occurrence — over ongoing debates about the direction of the project and its leader or 'maintainer.'

It is this small project that forms the 'what is' of this utopian study, a 'what is' that may also contain something of what 'ought to be.' The greater part of this study, therefore, will be in coming to understand how the 'what is' of Compiz is practically made and remade every day, to understand the processes and relations that perform its order. As we shall come to see, the performance of this order is at once social and technical, produced as much in the relations between people as in the relations between the objects of its virtual space.

§I went into this research with no clear methodology and what I ended up undertaking was a strange and distanced kind of participant observation. Strange and distanced because I was neither a participant and nor was my observation contempora-

neous with the events as they unfolded. As to the 'participant' part, I sought to familiarise myself with the tools and objects of the project, knowing full well that these objects would be crucial to any account of order. Immediately prior to and during this study I undertook learning three different programming languages, namely PHP, Javascript, and the predominant language of Compiz, known simply as C. I must admit to gaining an immense amount of satisfaction and enjoyment from the learning of these languages and in using them to write a number of small programs. I came to appreciate and understand some rather technical qualities to the writing of code and the construction of a program, notions such as modularity that I might have missed had I not had this experience. But I also came to appreciate so many of those 'subjective' qualities that made for much of the discussion on the mailing list, desirable qualities such as clean, beautiful or obvious code, or, as I discovered in many of my earliest programming attempts, 'spaghetti code' that ended up being utterly unmaintainable. I also sought to learn a few other key objects, notably the CVS revision system that contained the Compiz code and the different security models in place that would later be so crucial to the construction of virtual space. These endeavours culminated in a four week collaborative coding project to build a rather elaborate website management system and migrate a substantial dataset, wherein I discovered (and struggled with) the great difficulty of maintaining cohesion amongst fellow coders with whom face to face contact was impossible. The delegation of maintaining order to a number of virtual artefacts and documents during this project was fundamental to our group cohesion. So, it is in this strange sense that I am claiming to have engaged in the 'participant' part of the 'participant observation'

method, to have got a feel for much of the work, the everyday practices and the various artefacts even though I never participated in Compiz itself.

To the second part of that method, I think I can make a more robust claim to have observed the project, even as the events of this study had concluded some three years prior. With the exception of the fleeting utterances of the 'IRC' chat and private emails, the sum of communications between the collaborators of the Compiz project remain extant to this day. The full mailing list archives were available, the full history of the code iterations and commits were still present in the CVS repository, and the iterations of the related websites and discussion forums were available as 'snapshots' via the Internet Archive. My observations of the project were delayed, and occurred with some degree of foreknowledge of events, but proceeded roughly in sequence as they occurred.

The principal source of data for this project was, without a doubt, the Compiz mailing list as this was both the richest source of information and the heart of communication for the project. My methodology for observation, therefore, consisted principally of reading every email on this list for the period beginning in April 2006 until mid February 2007, totalling at just under 1500 individual correspondences. As emails pointed elsewhere, where they made reference to a code commit or to a website, to one of the forums or to an engineering standard, I attempted also to sight these other artefacts. My method was thoroughly ad-hoc. I generated a type of index of the emails, referencing major events and making note of exchanges that I thought to be interesting coupled with brief notes of my own. Additionally, I maintained a

diary of my thoughts as I went through the emails and as I read a number of loosely related published materials.

It was via these two methods — the detached participant and the late observer — that I came to create an order in my own mind of the ordering processes at work within Compiz and could begin writing and researching the three machines upon which I came to settle.

§This study can be divided into three main parts. To the first chapter is given the task of building up a picture of the Compiz project, in which we meet some of the different collaborators, come to observe many of its everyday practices and, crucially, are introduced to some of the objects of which it is composed. Chapter One is intended to give both a feel for the project and its many complexities, as well as provide something of a timeline leading up to the event of the fork.

Chapter Two sets out to lay the theoretical foundations for this study. Its task is primarily to develop the concepts of the 'machine' and the 'abstract machine' that are the principal means though which this study attempts to delineate the different generators of order within Compiz. This order, the 'what is' of Compiz, is not considered here as simply given, or else as something that is achieved once and for all. Rather, the emergence of social order requires an explanation, and where order endures we need to account for the mechanisms that give rise to its duration. Briefly, a machine is a set of thoroughly heterogeneous elements — the body, language, objects, spaces — that produce certain effects and orderings by way of these elements' connections to one another. By this model, the source of order lies neither in a

transcendent realm of 'structure' nor in an element's functional role within an organic whole, but instead arises immanent to the movements of material substance. It is the development of these two concepts — the machine and its abstract form — and their particular application within the virtual realm that forms the task of Chapter Two.

The three subsequent chapters each detail one of these machines. Chapter Three explores the abstract machine of the Passport and its instantiation within Compiz in the user-space machine. This machine works to regulate access within the virtual spaces of Compiz and produces a role known as the gatekeeper, one who may exercise a power both 'vicarious and precarious.' Chapter Four unveils the counterweight to the Passport known here as the Exodus. The Exodus is an abstract machine that makes the threat of desertion a real and ongoing possibility, which lays the groundwork for spaces elsewhere that are both plentiful and known, and in this establishes an 'imaginary counter-power' within the group. The abstract machine of the Exodus is instantiated within Compiz as the machine of the fork, and for the majority of this study it served to undermine the power of the gatekeeper and recast him as a steward of the code, as 'maintainer.' Finally, Chapter Five introduces the abstract machine of the Module, a machine whose primary aim concerns the spatialisation and organisation of the code, but which produces a concomitant spatialisation and organisation of developers and projects. It is this machine which produces the large scale order between free and open source projects, an order which can be characterised as a kind of anarchist federalism.

§This study is not meant to be read as a simple advocation of the organisation found within Compiz. Indeed, at least one of the machines present within Compiz — the Passport — appears wholly undesirable as a utopian model. Rather, Compiz, and free and open source software generally, offer us examples of types of human organisation, each differing in their desirability. Unearthing such microcosms of utopia is, therefore, only the first part of this utopian methodology. For in coming to understand these models, in coming to perceive what is desirable about them, we must also seek to understand those elements that are undesirable, those which are potentially dangerous. Indeed, H.G. Wells wrote that 'the creation of utopias — *and their exhaustive criticism* — is the proper and distinctive method of sociology' (Wells, cited in Levitas, 2005; emphasis added). This critical aspect is something to which we shall return in the Conclusion, but for now let us begin.

I. The Exploration

In which we meet the main protagonists, glimpse upon the realm, and embark upon our story

We start in the midst of things, on a mailing list for a project called 'Compiz.' It is 27 March 2006, and the first email appears baffling:

> Hi!
>
> Here are 2 patches for compiz:
>
> "compiz_show_desktop.diff" adapts metacity's show-desktop-behaviour, i.e. when compiz is in show-desktop-mode and a new window is opened or a window is maximized, only this window will be shown.
>
> "compiz_switch_all_windows.diff" will show every window in the switcher, not only non-minimized ones.
>
> Beware, I'm not really a C-coder, so things might not have been done the way they have to be…
>
> Thanks,
>
> Alex
>
> (Jasse, Alex: 2006-03-27 12:02)

Figure 1. Compiz-as-program displaying cube rotation between desktops, transparency, and shadowing of windows.

To which comes the reply, from a David Reveman:

> I've updated compiz so that show desktop mode works better. It's more like metacity's behavior but not exactly as I'm not convinced metacity's way of doing it is the best. Let me know what you think.
>
> […]
>
> -David
>
> (Reveman, David: 2006-03-31 05:28)

Having started on the mailing list, we are immediately pushed elsewhere. Other objects appear: patches, Metacity, something called C, and Compiz itself. In these two emails we have been privy to an immensely complex exchange, and yet it exhibits a

certain nonchalance that betrays its complexity. Our first task is to pull this momentary exchange apart in an attempt to understand what has just occurred before us.

Let us start by taking stock of the objects we have just encountered, the first being this thing called Compiz. Compiz is two things. It is in the first instance a 'binary file': a dense string of ones and zeros that are largely unintelligible to humans. When properly enacted within a computer, however, this string of ones and zeros becomes an object that exhibits a regular behaviour, one that allows for a set of interactions, one that occasionally behaves in unexpected ways, and one that sometimes breaks altogether. This Compiz, that is, becomes a computer program. Its function is to bring '3D' capabilities to the computer desktop. It allows for windows to wobble as they are dragged across the screen, renders some as translucent so as to reveal the windows behind, enables shadows to be cast by different elements, and allows for the whole screen to rotate between desktops as if a cube (figure 1). This is Compiz-as-program, a type of compositing window manager, one of the first for the Linux operating system. For the moment, let us consider Compiz-as-program as a 'black box,' one whose internal workings are a mystery of ones and zeros but which when properly enacted behaves as an intelligible object: a computer program.

There is also a second Compiz. This Compiz looks altogether different: it is a series of files and folders, each file containing text in a strange language, one that appears to be a mixture of English, mathematics and Boolean statements. This language is known as C (figure 2). Whereas Compiz-as-program is a 'black box' containing a mysterious interior, this Compiz, Compiz-as-code, is

```
if (d->prop_xid)
{
   /* translate from frame to client window space */
   if (top_region)
      XOffsetRegion (top_region, -fgeom.left_width,
                                 -fgeom.top_height);
   if (bottom_region)
      XOffsetRegion (bottom_region, -fgeom.left_width, 0);
   if (left_region)
      XOffsetRegion (left_region, -fgeom.left_width, 0);

   decor_update_meta_window_property (d, theme, flags,
                                      top_region,
                                      bottom_region,
                                      left_region,
                                      right_region);
   d->prop_xid = 0;
}
```

Figure 2. An excerpt of code from Compiz-as-code

simply a surface, its contents being immediately apparent. This is the 'source code' for Compiz-as-program, it is its blueprint, and it is the object upon which the entirety of the Compiz project's construction effort is made. This construction effort principally involves the reading and writing of this code. While Compiz-as-program is probed, studied for both its predictable and sometimes unpredictable behaviour, Compiz-as-code is instead subject to interpretation and editing. Moreover, these two Compizs are related. As a type of blueprint, Compiz-as-code can be subject to a process known as compilation wherein its designs and prescriptions are deployed to produce the object Compiz-as-program. In this process the source code, the surface that is Compiz-as-code, is passed through a compiler — a separate program — which outputs the unintelligible ones and zeros of Compiz-as-program, the binary blob, ready to be enacted by a computer (figure 3).

Let's continue to take stock of the objects we have encountered. There is the language in which Compiz-as-code is written,

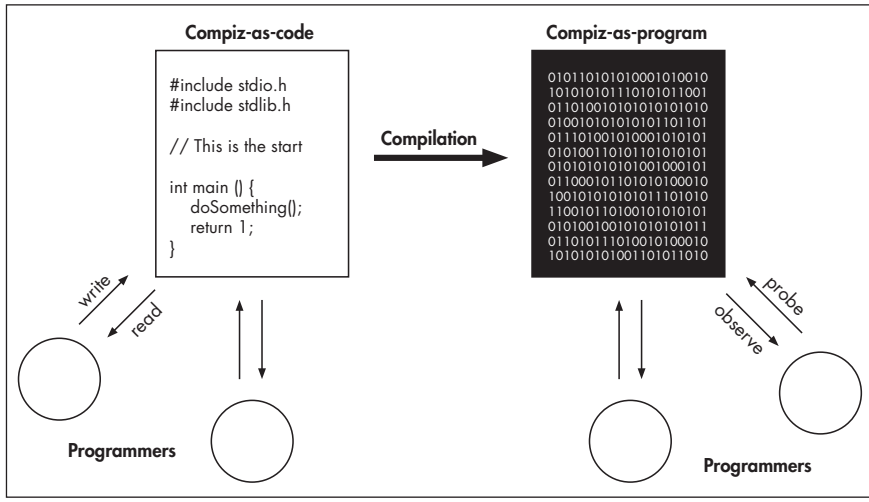

Figure 3. The relationship between Compiz-as-code and Compiz-as-program, and the different interactions that each allows with programmers.

'C.' C is a formally specified and standardised language, complete with grammatical rules, an allowed set of words, and so on. Dating from 1972, the creation of the language emerged out of previous attempts at developing programming languages, with each tuning a number of different language-specific parameters such as the expressivity of the language, the abstraction from the hardware, the language portability and the choice of control structures (Richie, 1993). Its technical implementation and rules are beyond the scope of the discussion here, but it is important to grasp one important aspect of computer languages like C. Programming languages, far from being incomprehensible and opaque machine code, are deliberately designed to 'lend' themselves to being understood, a property known as their expressivity. That is, the coding language aims to *communicate* its workings to those that are privy to its source code. A mixture of simple mathematical symbols and language-like statements are deployed to convey the workings of the code. This is aided still further

```
[…]
diff --git a/src/window.c b/src/window.c
index 81485f3..7b87a29 100644
--- a/src/window.c
+++ b/src/window.c
@@ -1316,8 +1316,9 @@ addWindow (CompScreen *screen,

w->frame = None;

-       w->placed = FALSE;
-       w->minimized = FALSE;
+       w->placed = FALSE;
+       w->minimized = FALSE;
+       w->inShowDesktopMode = FALSE;

        w->pendingUnmaps = 0;
[…]
```

Figure 4. An excerpt from a patch, showing the area to be added, deleted, or amended to the original file.

by the inclusion of plain English 'comments' which recur periodically throughout the codebase. Comments have no purpose within the program itself but rather function much like Post-it notes within the code and are intended to directly communicate with other programmers by explaining what tricky bits of code 'are doing,' the reasoning behind coding one way as opposed to another, remarking on the quality of bits of the code (especially where 'hacks' are used and are in need of future revision), and so on. The source code, therefore, is not *just* a technical object, but is also both highly expressive and linguistic.

Attached to the first email are two files that end with the suffix '.diff,' known as patches. A patch is a small file that, as its name suggests, is used to amend a much larger body of code. A patch is a file that describes only the differences between two pieces of code, such that a patch may be applied to one to transform it into the other (figure 4). They are especially useful for describing the changes made to a file compared to an earlier version as in

this instance, for example, where Alex has sent two patches which describe the changes she has made as opposed to sending the entire codebase. This has several advantages. It means that emails are lightweight, containing only the code changes. It also means that multiple patches from different people can be applied to a single codebase accumulatively, fostering collaborative development. But it is also a means of communication which, like the C language, lends itself to being read and makes it particularly easy to identify changes that have been made. The patch, therefore, is another kind of dual artefact, one that can be enacted as both technical and linguistic.

We have taken stock of the two different Compizs, the C language, and the patch. Let us return to the email exchange and make an attempt at constructing a story of what has occurred. Our original author, Alex, has been using Compiz-as-program, exploring its behaviour, probing its mysterious interiority. Alex appears also to have used Metacity — a widely-used *non*-compositing window manager — and to have similarly explored its behaviour, indeed to such an extent so as to have noted explicit conventions of behaviour. In this exploration she has discovered a discrepancy in behaviour between the two programs, and has sought to bring the behaviour of Compiz-as-program into line with the more popular Metacity. Additionally, she has noted some functionality that is missing in Compiz-as-program which she would like to have included. She has then moved from Compiz-as-program to her own copy of Compiz-as-code and embarked upon a substantial interpretative effort, seeking to make sense of the code and understand its workings, even in spite of 'not really [being] a C-coder.' The ability to gain access to the code in this way is unique to free software. Having edited the code, she has

used a tool known as 'Diff' to create the patches and has attached these to her email, an email which also includes the rationale for her changes. Finally, she has located the Compiz project mailing list and sent both the email and patches to the list in the hopes of inclusion into the official Compiz-as-code.

And what has been the response? David Reveman has read through the patches and has sought to understand the changes made therein. He too seems to be aware of the conventions of behaviour established in Metacity though he partially rejects them, claiming he is 'not convinced metacity's way of doing things is the best.' Similarly, he amends the second patch that Alex has submitted so that it 'just show[s] an icon for windows that are not mapped.' Both are subsequently applied as we can see in the revision logs for Compiz-as-code: '2006-03-31: Fix up show desktop mode and minimize' (Reveman, 2006a).

§We must pause and consider the nature of my description thus far. For what is a discussion of a brief email exchange, an apparently 'social' interaction, I have spent a great deal of time documenting instead objects and 'things,' the material artefacts of this exchange. Yet, it is things and things alone that make this exchange possible at all, an exchange where every moment is thoroughly mediated by a world of objects, including the emails themselves. *Mediation*, from the Latin mediari, 'to intervene, mediate' (Harper, 2001), is to alter a course of events in some way. Our objects are mediators in the full sense of the word: they do not simply transmit forces unchanged, but transform these forces, they intervene upon them. This concept of mediation forces the material artefacts that previously adorned the social background

to the fore, becoming fully commensurable to our humans. If we are to describe a course of events, we must pursue the flows of force through both our humans and the host of non-human mediators, pursuing these forces as they are bent, twisted, rebuffed or otherwise transformed by the life of objects (DeLanda, 1997; DeLanda, 2006; Latour, 2007).

When we talk of objects, of things, we may be led to think of them in terms of their function, as if their use is immediately apparent and they are simply 'used' by our humans, that they are mere elements in simple cause-and-effect chains. But objects are without essences, without transcendental qualities. I have used the term enaction in my description as a shorthand for the process whereby objects are made to produce certain effects by association with other objects, effects — sometimes unpredictable — which arise not out of the essential qualities of these objects but in the interaction itself. This forces us towards two conclusions: that objects may behave differently under different circumstances, and that objects are not static elements but are progressively revealed through processes that act upon them. These points, which I mention now in passing, shall undergo a full elaboration in the next chapter.

§An email and a patch arrive only the next day, 1 April, from Mike Hearn. He writes,

> I've started to pull my way through Quinn Storms differential, hopefully we can pull some stuff of this upstream.
>
> This one seems like a good place to start.
>
> Credit to Quinn Storm livinglatexkali@gmail.com

> ChangeLog:
>
> *plugins/gconf.c (gconfGetValue): Fix typo that caused color parsing to be incomplete.
>
> (Hearn, Mike: 2006-04-01 08:28)

Once again, let us try to understand the brief contents of this email. As before, we have the submission of a patch intended for inclusion into the official Compiz codebase. We also have an explanation in the email of the function of the patch, which in this case is to simply correct a typographical error in the code. The really interesting aspect to this email, however, is the source of this patch, for it has not come from Mike Hearn himself. In this case, Mike Hearn has not studied Compiz-as-code, nor moved between it and Compiz-as-program as we guessed in our last exchange; in this case, he has instead studied a different codebase known as QuinnStorm.

This mention of 'Quinn Storms differential' is the first mention on the mailing list, though it appears to have been in existence for some time already. QuinnStorm is a near replica of Compiz, a 'branch' of the official codebase that exists elsewhere, maintained not by David Reveman but by a person who goes by the alias 'Quinn Storm.'[2] The QuinnStorm branch exists by virtue of the permissive property relations of Compiz, and of free software in general, whereby code may be freely duplicated and modified, with the restriction that this duplicated or modified code be subject to the same license. We shall discuss further the nature of these licences in Chapter Four, but suffice to say for

2. 'Quinn Storm' — two words — shall denote the person, whereas 'QuinnStorm' shall denote the code. This is in keeping with the usage on the mailing list.

now that it is this permissive property regime that has allowed for the QuinnStorm branch to come into existence.

QuinnStorm started out as a simple duplication of the Compiz code, but has since had changes made to it by a number of people including Quinn Storm herself and at the time in which we find ourselves it has grown to incorporate a number of changes not present in Compiz. But it is not wholly independent. Compiz is, as Mike Hearn notes, 'upstream' from QuinnStorm. 'Upstream,' and its converse 'downstream,' are indicators of space, of a directionality of code flow. In any one project there will often be code that has been integrated from elsewhere, a bit of code from another project, for example, to handle drawing. Our project would then be considered 'downstream' from the drawing project. Perhaps, in the course of working with the drawing code our project were to find a bug, a problem in the code that our project has subsequently fixed. They could, and are usually expected, to pass such fixes back upstream. And similarly, our project may be included in a much larger project, as part of an entire operating system perhaps, which would be considered downstream. It is in this sense that Compiz is upstream from QuinnStorm. At this particular time we find that QuinnStorm was regularly synchronised with Compiz to keep the code aligned where possible whilst adding features of their own, and in this sense we can say that changes to Compiz are travelling downstream to Quinnstorm.

These upstream and downstream processes are not symmetrical, however. In his email, Mike Hearn's patches are designed to send code upstream and lessen this asymmetry. He has examined, studied, and interpreted 'Quinn Storms differential,' which as we may now guess is simply a particular output that details the sum

differences between the two codebases, and upon identifying one particular difference that he believes worthy of inclusion he has created a patch to be applied to Compiz-as-code. It is later on that day that we receive an email from David Reveman, noting simply, 'good, thanks […] done' (Reveman, David: 2006-04-01 14:26). In the revision logs to Compiz-as-code we see that the patch has been applied, '2006-04-01: Fix typo' (Reveman, 2006b).

At this point it is not yet clear, at least from the public documents, why QuinnStorm exists as a branch to Compiz, nor precisely what the relationship is between the two.

§We have seen a couple of emails from David Reveman, and we may now have guessed that he is someone rather quite powerful within Compiz. In both instances, he has been the gatekeeper to accepting code into the official codebase. It has been only upon his instigation that code has been committed, and in our first example he even substantially modified the submitted code. He is made powerful, constituted as powerful, through the control of the official codebase. On 4 April he sends an email to the mailing list stating explicitly his relation to the code:

> I'm currently maintaining the main compiz code, the gnome decorator and the set of plugins in CVS. Bug fixes and new features are much appreciated but I'd like to review all patches before they go into CVS.
>
> […]
>
> If you got a plugin or decorator, I'm more than happy to put it in CVS and give you commit access as long as you're willing to maintain it, there's a configure script

> option to disable it, and it's not a complete piece of crap.
>
> Send your patches to the list and I'll deal with them as soon as I can.
>
> Thanks,
>
> -David.
>
> (Reveman, David: 2006-04-04 05:32)

We have met another object worthy of investigation. CVS, an acronym for Concurrent Versions System, is a program that manages code. I have talked previously of revision logs when patches have been accepted, but it is actually CVS that has been recording these revisions. To understand how CVS is deployed, let us consider coding without it. Much like writing any sort of document, one would start with an empty document window and simply start writing code, saving one's work periodically and always presented only with the most recent revision. When there is only one person working on the code this is a possible method of coding. When there are multiple people working on the same codebase, however, things become much more complex. How do my collaborators know that I have made a change to the code, and how do they know specifically what changes I have made unless I also send them a patch? And if the changes I make turn out to be a regression of functionality, how do we 'undo' the changes I have made? How can I experiment with an idea without putting the code to ruin? CVS was created to address these problems by creating a log of all the changes or 'commits' made to the central code repository, where each commit contains a record of the date, the author of the new code, and a record of the change itself in the form of a patch. Additionally, a 'snapshot' of the code is made, allowing for any previous state of the code to be viewed in its

entirety or even, if it is deemed necessary, for the entire codebase to be reverted to a previous state. CVS is a kind of container for the code and lends itself to being enacted both in a roughly functional sense by managing the code, and in a communicative sense in allowing itself to be 'read' for changes to the code. Both of these make it easier for multiple collaborators to work on a single codebase.

In his email, David Reveman not only speaks of CVS, but also of giving particular people 'commit access.' In this we come to understand something else about the way in which CVS is enacted. CVS is a gatekeeper, but it is a strange one. It allows for code within its control to be freely duplicated and taken elsewhere: that is, it allows for code to go out, but it closely restricts, controls or else denies code coming in. At this present point only David has commit access to the official codebase, only he can submit new code, and all patches must pass through him. His suggestion, however, is to give interested people access to designated but limited portions of the code to which they will have commit access, and to whom is given the responsibility of maintaining that code and ensuring that it is not, and does not become, 'a complete piece of crap.'

If David is constituted as powerful through his control of the official codebase, then it is in the careful enaction of the object CVS as gatekeeper that this control is practically realised.

§There is no differentiation between topic areas on the Compiz mailing list, whether these emails concern areas avowedly technical, political or otherwise. Within a single hour we may see an email submitting a patch to improve the blurring algorithm

from one person, another writes that they have discovered a bug in a particular plugin, an email arrives that continues a heated exchange about the direction in which the Compiz project is going, and a final email arrives in response to the blurring patch claiming that it does not properly conform to some already accepted standard. And so, even as Quinn Storm continues to work on her increasingly differentiated branch, as David announces that people may apply for CVS access and, as we shall see later, dissention brews over the organisation and direction of Compiz, the practical task of coding continues. On 6 April, Mirco Müller, who has previously committed to writing a patch to provide 'tweakable drop-shadows,' writes,

> Greetings everybody!
>
> I started looking more thoroughly at gnome-window-decorator.c and now my head spins and "hurts" and believe that I'm not going to achieve anything serious in terms of tweakable shadows anytime soon. It's far more difficult than I expected. […] I'm currently looking like a jackass and feel just dumb for not really comprehending the code :/
>
> Best regards...
>
> MacSlow
>
> (Müller, Mirco: 2006-04-06 05:38)

To which Mike Hearn replies within the hour,

> [You are hardly a jackass.] :) The code is lacking comments, and I've had a hard time figuring parts out too. One thing I'd like to do at some point is go through one of the plugins and add some detailed comments explaining what each part does. […]

> For instance it took me a little while to figure out the animation scheme used […] Likewise the screen grab API isn't too hard to understand when you figure it out but [in my humble opinion] it's otherwise not obvious that "if (ss->grabIndex)" means "if the animation is currently in progress".
>
> thanks –mike
>
> (Hearn, Mike: 2006-04-06 06:28)

In this momentary failure to comprehend the code we are presented with a glimpse into the practical work of coding that is often largely hidden, subsumed under that general category of learned and routinised practices of technique. What resources is Mirco Müller seeking to draw from so as to proceed with his nominated task? Let us presume that he has already probed Compiz-as-program and found it lacking in the specific functionality of allowing for window shadowing to be customised. He has then moved to Compiz-as-code and attempted to understand the working of the code. He does not, however, seek to read and understand the totality of code. Rather, his work is reduced to interpreting and comprehending just one file within the codebase, gnome-window-decorator.c, work that is made possible by the particular organisation of the code into different sections. He thus appears already familiar with the common technique of separating out different functional elements of a codebase into separate files. Additionally, he approaches the codebase already practiced in the techniques of reading and comprehending the C programming language. Even so, despite his efforts and his stock of technique, the code remains before him as a problem to be deciphered, one requiring significant interpretive energies, and it

is because of this that the process is made explicit on the mailing list.

Mike Hearn's reply points to a number of failings of the code that make it difficult for it to be read for its workings. In the first instance, the code incorporates few comments. These plain-English remarks that reflexively comment on the workings of the code are missing, thus forcing Mirco Müller to rely entirely on the programming language itself. We can see the importance of properly commented code in Mike Hearn's suggestion to 'add some detailed comments explaining what each part does' of a particular plugin. This would not be because the chosen plugin would be especially important but because it could be used as a template, deploying its similarity to other Compiz plugins so as to make them, too, more easily comprehensible. For Mirco Müller, however, the lack of comments is made still more difficult by the idiosyncratic style of coding currently in place. While the C programming language is in a sense a strict set of grammatical rules, the substantive content of the code is the prerogative of the programmer: the names of functions can be entirely arbitrary, the methods through which certain functionality is 'exposed' can be as complicated or as simple as they like. The code in gnome-window-decorator.c is not easily read precisely because the function names chosen are not particularly descriptive, and much of the code is indirect and allusive as Mike Hearn makes clear in his example.

Later on that day we receive an email once again from Mirco Müller reading,

> After further investigation my head spins the other way around :) In the meantime I was able to identify

> all drawing functions responsible for the titlebar (I didn't touch the title and buttons), frames and shadow elements. As an example I replaced them with simple opaque rects to see what part goes where.
>
> (Müller, Mirco: 2006-04-06 10:37)

The email continues, describing a number of unexpected effects of changes to the code. We are privy to another technique of coding, that of simple experimentation: Mirco Müller is here altering aspects of Compiz-as-code, compiling it into Compiz-as-program, and proceeding to probe this latter object for changes, if any. In this back and forth motion he seeks to uncover the relation between the two objects, and ultimately to develop an understanding of Compiz-as-code so as to purposely make changes in the behaviour of Compiz-as-program.

This technique of experimentation produces a number of unexpected behaviours in Compiz-as-program, and Mirco Müller returns once again to the email list seeking clarification, now directing his questions at David Reveman, the original author of gnome-window-decorator.c:

> Why did you do it this way David? It appears to be very non-obvious. Are there speed-issues demanding such an approach or other things going on behind the scene, which I still fail to see?
>
> (Müller, Mirco: 2006-04-06 10:37)

So begins a series of quite technical and verbose emails between Mirco and David, as Mirco produces more specific questions and David replies with his reasoning for the code as it stands. Mirco continues to employ the technique of experimentation with Compiz-as-code as he probes the resultant Compiz-as-program and, for example, on 7 April he writes, 'I can comment out that

portion [of code] and the whole drop shadow stays intact after I did a recompile /install and full restart' (Müller, Mirco: 2006-04-07 19:06). In the course of this exchange, David repeatedly describes the workings of Compiz-as-object in terms of smaller, functional objects, each sub-object interacting purposively with other sub-objects in their own constructed realm. One such description from 9 April reads:

> g-w-d puts all the quads that represent how decorations texture is mapped to a window in an X11 property on the client window. This property is read by the decoration plugin, XChangeProperty is use[d] for updating this decoration property.
>
> (Reveman, David: 2006-04-09 10:26)

The sub-objects of Compiz-as-program 'put' things in place, 'represent' aspects of their selves, 'read' one another, and 'use' one another toward desired ends; this conceptualisation of Compiz-as-program is prolific. Finally, on 10 April, David sends to the mailing list 'a small [incomplete] patch which adds some basic support for dynamic shadows to [gnome-window-decorator]' — to act as a kind of template — and he invites Mirco Müller to '[take] this patch and [fix] the last pieces so we can move it into CVS' (Reveman, David: 2006-04-10 07:45). Silence ensues on the part of Mirco, and on the 24 April an unannounced commit to CVS is made by David that reads 'Add configurable drop-shadows,' presumably implementing this feature (Reveman, 2006c).

In this exchange we get some sense of the work required for collaboration in Compiz, of the great difficulty even for experienced programmers to comprehend code. We have seen as well the techniques available to aid in this endeavour: the organisation of the codebase into smaller, functional files; the provision

of comments in the code; the design of the code that lends itself to 'obviousness'; the use of templates; and the back-and-forth process of experimentation with Compiz-as-code and Compiz-as-program. Though these techniques, with the exception of the last, have as their ultimate aim to ease the hurdles of collaboration, of people working together, they are once again not directed immediately at people but at objects; they are the moulding and shaping of objects with the vicarious intention to alter human trajectories.

§David controls access to the official codebase, but he does not control its designation as official. His relation to the official codebase is not the relation of exclusive ownership that is the mainstay of capitalist property relations, but is rather a kind of stewardship over the code and over the project. He is known as the 'maintainer,' a word which captures well the precariousness of his position: should he fail to maintain the codebase properly, should he become difficult to work with, then he will simply be bypassed, the code will be duplicated and taken elsewhere, and a new project will be formed. This process is known as 'forking.'

Mirco appears again on the mailing list on 25 April submitting a patch continuing with his interest in customising the window shadowing,

> Greetings everybody!
>
> Here's a patch (against Compiz from CVS-head about 30 min. ago) that adds a shadow_color option to the parameters of the decoration plugin. I only needed to

> add a few lines to decoration.c and gnome-window-decorator.c.
>
> (Müller, Mirco: 2006-04-25 11:28)

Three days later on 28 April David replies,

> Patch looks OK, I'll add it if people think this functionality is useful. I don't want to add options just because we can.
>
> (Reveman, David: 2006-04-28 04:16)

The reluctance on David's part to proceed with committing the patch to the official CVS is unmistakable. Yet it is David that must be convinced if Mirco is to have his patch accepted as part of Compiz proper, and it is this issue that sparks a debate regarding David's style of maintainership. On the same day as David's reply, Quinn Storm writes to the list in defence of the inclusion of Mirco's patch, to which David replies some time later on 3 May,

> We can expose all kinds of useless crap through options if we want. If no one uses an option except for when trying what it does, then it's useless. I don't want useless options. […] To me, the shadow color is not an obvious thing that people want to adjust.
>
> (Reveman, David: 2006-05-03 03:38)

Mirco appears absent in this continuing debate, but Quinn Storm replies the same day,

> I've gone ahead and applied this in my CVS, I was waiting to see first if it was going to be applied upstream but it appears unlikely that that will happen.
>
> (Storm, Quinn: 2006-05-03 20:39)

We may now infer at least part of the reason for the existence of the Quinnstorm branch. Namely, that it exists to incorporate functionality that has been rejected or is otherwise missing from

Compiz official. It provides a space, that is, in which to organise differently. In another email on the same day, Quinn Storm asks,

> Who gets to make the determination as to what is "useless crap"?
>
> […]
>
> It's beginning to look like Compiz will fork early in its development, one toward configurability and options, the other toward your vision. I wish this did not have to happen, and hope it does not.
>
> (Storm, Quinn: 2006-05-03 09:39)

The threat of a fork is serious indeed. While forks are allowed for by the permissive property relations of free software, they remain rare events. A fork is a collective endeavour. It is the realignment of a number of developers' and users' allegiances from one project to another, the shift of officialdom. Quinn Storm's threat of a fork is not an empty threat, for it is the Quinnstorm branch of code — which includes additional patches and functionality — that is most commonly used downstream by the major Linux distributions.

A similar episode follows some time later on 19 June regarding a number of patches to implement Xinerama support, a form of multi-screen support. David rejects the Xinerama patches, and promises to implement 'proper multi-screen support' himself (Reveman, David: 2006-06-19 06:25). Less than 20 minutes later, Colin Guthree replies, 'Until David has completed the "proper" multiscreen stuff[, t]he Quinn CVS version of compiz has Xinerama support' (Guthree, Colin: 2006-06-19 06:49).

On 24 June, and amongst escalating tension amongst the developers, Guillaume Seguin writes to the list,

> lots of work has been done outside of the official team by the new Compiz community […]. These unofficial developers are mainly using the compiz.net [bulletin] boards, #xgl or #compiz-dev IRC [chat] channels on Freenode […]. Most patches written by these developers get committed to Quinn Storm's cvs […]. Nevertheless, it seems that unfortunately little of the very good work that is done actually gets into the official project, which can make it difficult to continue to be enthusiastic about developing for Compiz.
>
> (Seguin, Guillaume: 2006-06-24 16:24)

Guillaume proceeds to ask of 'the standards that any plugin or patch must meet before it can be included in the main codebase,' and asks whether the reasoning behind patch rejections could be provided to the mailing list when these reasons are for issues other than simple quality. Finally, he concludes by asking 'if it'd be possible to discuss with you a bit more of what we're doing, what you are doing, and what we should do to help.' Though never stated, we can assume the 'you' to whom Guillaume's email is directed is David Reveman. He does not reply.

It worth noting in Guillaume's email the mention of other places where work on Compiz is happening, places beyond the official domain of the mailing list that include both a bulletin board and chat rooms, and the Quinnstorm branch itself. The vastness of space coupled with the ease with which certain objects may be duplicated in the virtual realm is fundamental to allowing contesting modes of organisation to form. This is to say, the contestation over the organisation of Compiz does not have to happen directly, it is not restricted to contestation within the single nexus of the Compiz project, but may occur instead by a simple aban-

donment of the centre and the concomitant production of space elsewhere in which to work.

This space elsewhere is maintained for the next few months. The Quinnstorm branch becomes progressively differentiated from the Compiz branch, as the developers involved in these other spaces write and submit patches that are not applied to Compiz official. On 15 September, Shawn Starr writes to the mailing list seeking to 'get a better understanding as to why Quinn's patches have not been accepted into the Compiz git tree' (Starr, Shawn: 2006-09-15 12:03). Though he concludes that 'it would be best to avoid a fork if possible,' the Quinnstorm branch is no longer simply a slightly modified version of Compiz, and in many respects the two branches have become separate projects already: there is limited developer overlap between the two branches, the codebases are significantly different, they are working towards different ends, and they occupy different spaces. Quinn Storm replies the same day regarding the nature of development within Compiz,

> In general, it at least 'feels' as though development is rather closed, with any possibility of getting code into the main source tree being at best a procedural headache.
>
> […]
>
> In the end, I think I'll let the statistics speak for themselves. Most people using compiz are using the community compiz tree [ie. Quinnstorm], or packages made from it.
>
> I don't want there to be any animosity, but perhaps our ideas of the direction for this project are simply too dif-

> ferent. If that is the case then it would simply be in everyone's best interest to have an amicable fork.
>
> (Storm, Quinn: 2006-09-15 15:07)

Finally, on 23 September we see an email from Colin Guthree asking, 'I don't know how much of a political hot potato this suggestion will be […] would you consider adopting the csm plugin from beryl into compiz?' (Guthree, Colin: 2006-09-23 01:12). This mention of a project called Beryl is the first indication on the mailing list of the fork made official, constituted as separate by the choosing of a new name. In fact, it was five days earlier on the 'community' bulletin board that the fork was announced (Seguin, 2006).

§Somewhat tongue in cheek, in their study of organising in open source projects Lanzara and Morner note that these programmers 'basically do two things: write programs and have e-mail conversations about programming' (2005: 69). In the preceding passages I have attempted to provide a series of glimpes into the mundane and everyday practices of collaboration within Compiz which, indeed, consists principally of these two types of practices in addition to a significant amount of interpretive work. But such a description of this project as simply writing programs and having email conversations, as both Lanzara and Morner argue, entirely misses the mediation and work performed by the vast array of objects, artefacts and spaces that combine to produce the character of organisation within Compiz. It misses the role of the code itself in communicating amongst developers, in the role of the CVS repository in gatekeeping the code or the effect of the vast spaces of the virtual realm in undermining the

official status of Compiz. I have here, therefore, introduced some of these objects and, additionally, provided something of a general history of the project over the course of several months until its eventual fork. Having jumped in at the deep end and given a 'feel' for the life of the project, it is to the remaining chapters to describe, firstly, the means through which we shall attempt to understand the ordering of Compiz, and then to undertake a description of the three primary machines operating within the project.

II. The Abstract Machine

On the problem of order, and its production in and through things

The natural sciences conceive of the cosmos as undergoing an irreversible decline, a terminal fall from a state once highly ordered to one increasingly disordered. Those pockets of order that buck this trend, with the greatest, perhaps, being the very emergence of life itself, arise only at the expense of greater disorder elsewhere and only through an incredible happenstance of processes. Order is that which must be explained and its emergence, however banal and commonplace it may seem, must be considered as something truly exceptional.

Social order does not escape these strictures. Its emergence and ongoing maintenance must be considered both commonplace and exceptional at one and the same time. The overlapping regimes of order that we have just observed within the everyday practices of the Compiz project *beg* explanation. This is not, however, because order is something that is alien to human groupings, that the otherwise natural social condition is a vicious war of all against all. Nor is it because this particular human association currently under study — Compiz — lacks the unitary prin-

ciples of the State or authority, that it lacks recourse to coercion and a political centre. Rather, order requires explanation simply because difference is base, because disorder is the cosmological 'state of nature,' and because when social order does indeed arise, as it tends to do time and time again, this is a feat whose achievement is truly incredible. The problem of disorder, that is, must be replaced with the problem of order.

The question becomes, how do we account for the rise of social order? This is the task of the present study, both to build a model that can capture the production of social order and apply this to the Compiz project. That model is the 'abstract machine' and its concrete corollary in the 'machine,' a concept born of the collaboration of Gilles Deleuze and Félix Guattari, and a model which I intend to complement with insights from Michel Foucault and Bruno Latour. In each of the subsequent chapters, we shall be analysing the organisation of the Compiz project by delineating the three most important abstract machines and their concrete instantiation: the Passport, the Exodus, and the Module. This chapter forms the theoretical prelude to those that follow.

§In developing the model of the abstract machine, we need first elaborate upon three ontological assumptions. The question of 'what there is' and its nature is intimately tied to any model that proposes to explain order. It is an ontology that, in the first instance, renders order either as problematic or as simply given and, in the second instance, delineates the resources at hand with which to provide an explanation. I am here making three ontological assumptions: that process and flux are ontologically base; that identity and order are the ongoing result of process and

constantly at risk of collapsing due to an excess; and, finally, that there is but a single ontology to which all things are immanent.

The first two of these assumptions are intimately linked. Deleuze asserted that 'it is difference that is behind everything, but behind difference there is nothing' (Deleuze, cited in May 2005: 19). This concept of difference, for Deleuze, was not the difference between established and pre-existing identities, but was instead something more fundamental, a 'pure difference' that was a process, an unbounded unfolding of substance prior to identity. This is an ontology that posits the world as always and already in process, that change and flux operate at the very basis of substance and that disorder, not order, is primary to the world. It is in this constant movement, in the ebb and flow of substance, that there arises identity, sameness and order. These *relative stabilities* emerge only as a result of ongoing processes that, with great effort, contain and direct — 'territorialise' — the underlying pure difference, a pure difference that threatens always to overspill from within (May, 2005: 128). Being is illusory or, rather, *being is becoming*, and identity is always at risk of rupture (May, 2005: 60).

This ontology has been described as in keeping with the Heraclitean tradition, and for much of the history of Western philosophy it has been relegated to a subterranean existence (Graeber, 2001: 50). That which dominated can be traced back to Parmenides, a tradition that held objects as ontologically base, where the building blocks of all things were these fixed, static, and unchanging elements producing an equally fixed, static, and unchanging world. Identity and order were quite unproblematic, and change was recast as illusory. In this formulation, becoming was mere appearance; all things were ultimately being.

The implications of the Heracliten ontology for social theory are twofold. Firstly, it is order, not disorder, which must be explained. Secondly, order is not something that is achieved once and for all. Unlike the Parmenidean tradition, wherein one could create order and expect that order to remain until affected by a source from without, the Heraclitean tradition recasts order as temporary, permanently at risk of decay by the underlying processes and excess immanent to a system. Order, therefore, is better described as *ordering*, as process and not state, being never finally achieved. Order, to borrow a concept from queer theory, is not ostensive but performative (Butler, 1990).

The idea of social order as performative is not new. It is, for example, one of the key assumptions of Garfinkel's ethnomethodology, literally the 'methods of people' in making sense of the world, of creating order in everyday encounters. As Anne Warfield Rawls wrote,

> The word 'Ethnomethodology' represents a very simple idea. If one assumes, as Garfinkel does, that the meaningful, patterned, and orderly character of everyday life is something that people must work constantly to achieve, then one must also assume they have some methods for doing so.
>
> (Rawls, 2002: 5)

Here, sense and order is something that must be reconstituted in each interaction with recourse to a set of shared methods, an order easily ruptured as was demonstrated in his numerous 'breaching experiments' (Heritage, 1984: 78–84). Bruno Latour, too, stresses that social order, as the outcome of ongoing relations, simply disappears when those relations disappear, that order doesn't have an 'inertia' or 'solidity' in and of itself. Arguing

against social theories that award order 'for free,' those he labels as 'sociologies of the social', he writes,

> [in these theories] the rule is order while decay, change, or creation are the exceptions. For the sociologists of associations, [however,] the rule is performance and what has to be explained, the troubling exceptions, are any type of stability over the long term and on a larger scale.
>
> (Latour, 2005: 35)

Thus for Latour, order is an achievement against a backdrop of disorder and its maintenance a process against decay.

The third ontological assumption holds that there exists but one plane of substance, and that all things are immanent to this singular plane. Deleuze, drawing from Spinoza, described this plane of immanence as *pure* immanence, '[it] is in itself; it is not in something, *to* something; it does not depend on an object or belong to a subject' (Deleuze, 2001: 26; emphasis in original). All things are expressions, unfoldings and refoldings of a singular, univocal substance, an unfolding and refolding immanent unto itself. There is no outside or beyond, no external cause to the world or to life; all processes must find their cause within themselves. Banished, therefore, are the ontological realms transcendent to the plane of the everyday and material world, whether these be God or the Ideal Types, History or Social Structure. Banished, too, is the ontological distinction between the non-human and the human. All things are continuous with one another. This is a radical materialism. But, to be clear, it is not one that gives special place or primacy, *a priori*, to a particular material domain, it is not one that privileges the engines of history in the means of

production, nor is it the creation of a structure and concomitant superstructure.

This final ontological assumption has several important ramifications. Firstly, that 'structure,' systems, order, and so on, must be accounted for without 'jumping.' We cannot explain away the orderliness of everyday life by claiming it to be an effect of a realm transcendent to our own, that by the interface of something akin to the pineal gland order is transported from the world of structure to the realm of the everyday. Latour, instead, insists we adhere to a 'flat ontology,' that we become 'myopic' and follow the transportation of forces bit by bit, following their material transformation from site to site (Latour, 2005: 165–172). It is important to stress that this is not an embrace of a social theory of pure localism. All sites 'local' are at one and the same time the provisional and moving terminus of a great number of forces travelling through space and time, travelling by means fully material and fully traceable (Latour, 2005: 196). That is, they are always and already constituted by forces originating from elsewhere. Moreover, global structure does indeed arise, but the onus is upon the sociologist to explain such order without transcendent worlds, by following wholly material transportations and transformations of force. When one approaches the question of large scale order whilst adhering to a flat ontology, the achievement of scale becomes something truly remarkable indeed.

The second consequence of immanence is that we embrace a radical anti-humanism and bring the background world of objects, spaces, and things both technical and natural, into the foreground alongside humans. Social accounts cannot consist of a world purely social, a world that unfolds on top of, but which

remains distinct from, the material. In our myopic tracing of connections between sites, therefore, we must include these heterogeneous objects in our accounts, moving in one moment from the human to the technical, the natural to the wholly synthetic. Indeed, it is precisely and only through this central role of objects that the achievement of scale becomes possible at all. Deleuze's 'univocity of substance' encourages us to go further, however, to embrace the vitality of objects or, as Latour writes, we must elevate the role of objects to full-fledged *mediators* and, if warranted, actors. The objects of our world, that is, do not function as mere intermediaries of forces, transparently transporting forces from site to site; indeed, this would be to relegate them once more the background, as mere 'things' upon which the social is writ. Instead, the objects of our world mediate, transform and operate upon these forces. Objects become *interesting*, they become essential in accounting for the ongoing production (or collapse) of order; their roles become transformative and, at times, entirely unpredictable (Latour, 2005: 63–86).

The final ramification is that we must now situate knowledge and give it material form. Ideas exist on pages, written upon hard disks, arranged amongst neurons; they are eminently material, and their transmission occurs only via material processes, only through the expenditure of energy, through the processes of reading and writing, printing and publishing, transportation and distribution. There is a tendency even for those orientations avowedly materialist to allow for ideas and knowledge to escape and become detached from the material on which they are bound, for them to gain an 'immaterial' quality. A materialist conception of ideas and of knowledge, such as this, is not to subject ideas to the material realm or to suggest that, in fact, ideas only 'reflect'

extant material practices. It is, instead, to suggest that ideas *are* material (Graeber, 2001: 54). This is especially important when considering the topic at hand, for the realm of the virtual, more than anything else, is most quickly and most easily detached from its material basis.

§To review, we are starting from the assumptions that order is the ongoing result of process and is generated in interaction, that this interaction is not just made of social 'stuff,' but consists of interactions between bodies, objects and spaces — things wholly material — and that order must arise immanent to these elements. These ontological 'clamps' set the foundation for building the model of order that shall be of employ throughout this study: the 'abstract machine' and its corollary in the 'machine.'

Let me attempt a first formulation. A machine is a heterogeneous collection of interacting elements that produce, in their ongoing connections to one another, emergent properties, effects and orderings. Its properties emerge not from the elements themselves, but in the connections its elements establish, in practices, relations, and interactions. Also known as an 'assemblage,' a machine is quite unlike its everyday namesake which appears to the world as static and whose function appears as simply given. This conception of the machine is neither static, as its properties only emerge in the course of movement, nor is its function simply given, wherein its effects can only be known *after* it is brought into combination with other machines. In her work on Deleuze, Claire Colebrook has contrasted these conceptions:

> In *Anti-Oedipus* and *A Thousand Plateaus* Deleuze and Guattari use a terminology of machines, assemblag-

> es, connections and productions […] An *organism* is a bounded whole with an identity and end. A *mechanism* is a closed machine with a specific function. A *machine*, however, is nothing more than its connections; it is not made by anything, is not for anything, and has no closed identity.
>
> (Colebrook, 2002: 56)

The machine is, in the end, a way of conceiving of 'wholes' that makes neither the mistake of reductionism nor functionalism. The movements of the machine come not from a simple aggregation of the properties of its smallest elements, a move which would be a return to a Parmenidean ontology, nor are its movements defined from without, either in relation to its place within a greater whole or its functional destiny; they emerge 'blindly' in and through the interaction of its various elements (DeLanda, 2005: 9–11).

This is a conception of a whole wherein the connections of its parts produce of each other their properties, properties that are not simply 'given' but are produced anew in and through each interaction. These 'parts' refer equally to objects as they do bodies. When we speak of bodies, however, the term 'properties' is usually replaced by something else: *subjectivity*. We can say, therefore, that in the intersection of the body and the plethora of machines to which it connects we find the ongoing production of subjectivity. This process is not the imposition of a certain type of being upon a subject whose essence desperately seeks something else, but rather is the very production of its desires, its knowledge, its movements and practices. Here there is no essential interiority, only surface. Such a conception immediately evokes Foucault's notion of the productive effect of power and its inscription upon

the body. Indeed, while we have been talking of 'connections' and 'relations' between elements, these terms can be replaced with Foucault's notions of 'power' and 'force relations' (Foucault, 1998: 92), wherein a machine is always and already a set of power relations.

The concept of the machine is ontologically mobile, in which machines are made of elements that are themselves machines (May, 2005: 122). The machine of the body interacts in concert with the objects of urban space to create the machine of the city, which itself operates in concert with the surrounding rural areas to form a (porous) bioregional machine. These machines, however, are not perfectly encapsulated within one another nor does each peacefully work in combination. Machines interact and overlap, they engage in ceaseless confrontations, transformations and recombinations, strengthening or destroying one another. 'Everywhere *it* is machines — real ones, not figurative ones: machines driving other machines, machines being driven by other machines, with all the necessary couplings and connections' (Deleuze & Guattari, 2004a: 1).

There is a second concept we need to introduce. Whilst the machine has a concrete reality existing only in one place, the *abstract machine* is instead a diagram that describes neither the specific elements nor the actual mechanisms of a particular machine, but the set of *relations* that produce the machine. Deleuze writes, 'the diagram or abstract machine is the map of relations between forces, a map of destiny, or intensity' (Deleuze, 2006: 32) and, citing Foucault, he writes 'it is a diagram, that is to say a "functioning, abstracted from any obstacle […] or friction [and which]

must be detached from any specific use"' (Deleuze, 2006: 30).[3] The abstract machine, therefore, describes a whole class of machines that share the same kind of logic, that enact the same kind of power relations, each of which is an 'instantiation' of its abstract form. The abstract machine is 'mechanism independent' such that the elements of a concrete machine may be interchanged with others so long as the connections remain of the same kind (DeLanda, 2002: 15). The concept, therefore, describes the ideal type of a class of machines, their pure form reduced only to their techniques of power and the connections of their parts. The abstract machine is also known in the work of Michel Foucault, in which it goes variously by the names the diagram, the general method, the modality of power, and the *dispositif* (apparatus).[4] Let us unpack these concepts by taking an example.

The Panopticon is the best known of Foucault's diagrams from *Discipline and Punish*. The Panopticon was a design for a prison that was proposed by Jeremy Bentham in 1785. It was an architecture where prisoners were to be separated into individual cells, the cells being arranged around the periphery of a circle at the centre of which was an observation tower. This tower contained a room upon whose windows were hung venetian blinds and whose entrances were concealed such that, from the outside,

3. In the same passage, Deleuze ascribes to abstract machines the ontological status of being real, existing immanent to substance, and as 'causing' the machines which take after them. This is not an aspect we shall be pursuing here.

4. Of the apparatus Foucault explains, 'What I am trying to single out with this term is, first and foremost, a thoroughly heterogeneous set consisting of discourses, institutions, architectural forms, regulatory decisions, laws, […] — in short, the said as much as the unsaid. Such are the elements of the apparatus. The apparatus itself is the network that can be established between these elements […]' (Foucault, cited in Agamben, 2009: 2).

prisoners could not see into tower and could not know when guards entered or left. The prisoners, however, could be seen at each moment, illuminated by the light shining through from the windows positioned behind their cells. The Panopticon was thus a surveillance machine that operated in one direction only — a prisoner could not know at any one time whether the guard in the tower directed their gaze toward the individual prisoner or if the tower possessed a guard at all. It was a reversal of the principle of the dungeon, from a condition in which the prisoner was not seen, was secluded and out of sight, held in collective confines with other prisoners and shackled with iron chains, to a condition of 'lightness,' of visibility, the fully individualised prisoner always already under the gaze of the tower. Foucault's primary thesis was that the Panopticon induced in the prisoner 'a state of conscious and permanent visibility that assur[ed] the automatic functioning of power,' a gaze that came to be internalised within the prisoner such that they regulated themselves (Foucault, 1995: 200-204).

The Panopticon was never built and yet it formed a central role within Foucault's analysis. Why, therefore, did Foucault introduce the Panopticon into his analysis when it was, as he later said, 'a utopia,' when 'all the history of the prison — its reality — consisted of having passed this model by' (Foucault, cited in Wood, 2007: 250)? The Panopticon-as-building certainly never existed but the Panopticon-as-diagram, as abstract machine, came to be widely instantiated within a number of disparate machines. The abstract machine of the Panopticon, Foucault argued, spread bit by bit, at first limited to a small number of sites but whose techniques became dispersed and were taken up, piecemeal and unevenly, across a variety of institutions — from the army to the

school, the hospital to the factory, as part of a generalised method of discipline. For Foucault, the Panopticon,

> […] must not be understood as a dream building: it is a diagram of a mechanism of power reduced to its ideal form; […] it is in fact a figure of political technology that may and must be detached from any specific use.
>
> (Foucault, 1995: 205).

The abstract machine of the Panopticon was a set of relations between certain elements: the cells provided for the individualisation of the prisoners, the combination of venetian blinds, the windows and the twisting and zigzagged entrances to the tower provided for the unidirectional gaze, and the arrangement of space provided for the possibility of the omniscient guards. Each of these elements, alone, possessed none of the qualities of the Panopticon, but brought together they came to produce in each other the effect of the individualising gaze of power. Moreover, the mechanisms employed in the concrete instantiation of the abstract machine mattered only insofar as they effected the appropriate relations; Bentham's Panoptical tower, for example, could just as easily be replaced — under the right circumstances — with a security camera or a computer log file perhaps.

Foucault's description of the Panopticon and its historical trajectory is one of the best and most fully articulated of an abstract (and concrete) machine in the literature. Its success, however, has come with a price. There has been a tendency since to treat the Panopticon in much literature as something rather unique, to see in more recent developments the same fundamental relations of the Panopticon. David Murakami Wood, for example, in writing about Foucault's legacy in the area of surveillance studies, suggests that the literature has confined itself largely to variations on

a theme, recasting modern machines as 'panoptic,' 'superpanoptic,' 'neo-panoptic,' and 'omni-panoptic.' Wood argues against this limitation, specifically regarding the technology of the database, saying,

> If Foucault had continued his genealogical historical account into the twentieth century, it seems unlikely he would have described databases as superpanoptic, rather he would have treated the 'database' as a particular political technology, a diagram, a mode of ordering, of its own space/time of power/knowledge.
>
> (Wood, 2005: 253)

The diagram of the Panopticon is just one diagram among many, a diagram that, even in *Discipline and Punish*, occupied a place alongside others lesser-known such as the table, the examination, and the carceral. This proliferation of abstract machines is something I intend to pursue in the course of this study: we shall come to trace out both the abstract and concrete relations of three distinct machines in operation as part of the Compiz project, treating them each as their own particular political technologies, each with their own set of relations, their own distinct effects.

§The elements central to the Panopticon — its towers, windows, venetian blinds, guards, centre and periphery, prisoners, cells — were fully heterogeneous, concerning bodies, objects and spaces. Objects and spaces, not commonly a focus of sociological accounts, are central to the operation of machines. As we have seen in Deleuze's immanent ontology, in Latour's emphasis on mediators in his social theory, and in the elements of Foucault's diagrams, it is bodies, objects, and their production of space that

produce the types of ordering we observe around us. We need in this final section, therefore, to consider the notions of objects and space within the virtual world of Compiz. In a nutshell, we shall be adhering to a strictly materialist ontology, letting at no point the 'virtual' escape the single plane of the material, that we shall consider 'virtual' space as real space, that we will be ensuring that the code — whilst no doubt a type of 'knowledge' — remains always conceived as inhabiting *some place* at *some time*, and that the practice of coding is understood not as a type of immaterial labour, but as the ongoing transformation of a series of objects. Let us take these points one by one.

In the first instance, let me at once stress the thoroughly mechanical and material basis of computer networks. A website or code repository is stored on a hard disk spinning at thousands of revolutions per minute, a hard disk which is contained within a server cooled by server fans, a server that is powered by an electrical cable that, perhaps, finds its terminus at a massive turbine which spins under the force of water trapped behind an imposing concrete dam. The network cables of that same website are maintained by a constant workforce digging trenches, burying cables beneath roads and footpaths, repairing broken linkages, eventually meeting in massive exchange terminals before diverting out once again. Throughout this, the first law of thermodynamics remains intact: this communication is not free, the spaces upon hard disks are vast but still limited, and at each moment energy and work is required for its upkeep. This is a return of our Heraclitean ontology: that which appears on the surface to be a kind of smooth space, a space of rapid transit and unbounded realms is in fact an alienation of sorts, one that hides a massive apparatus involving the expenditure of work and energy, one whose effect

is to sustain the apparent permanences of virtuality. This is not, however, to *reduce* the virtual to the physical and mechanical tools from whence it is derived, to suggest that the virtual is nothing more than these cables and such, for novelty does indeed arise in the interactions of these parts. It is, rather, to remind us that the virtual does not *escape* the material, that it does not stand in contrast and set apart from the world of the real. The virtual, that is, is a realm both novel and yet fully material.

That virtual space is produced, that it is in a sense an artificial creation, is not to somehow make it a pseudo-space, a space not quite real. Virtual space, like all space, is both real *and* produced. Space is not an abstract set of coordinates in which things happen, that mute and inert stage upon which life occurs. Space is produced when disparate machines and forces, each on their own trajectories, come into interaction. Space is precisely that place where things share a common existence in a common time. For all the divergent histories of those things, and the divergent becomings that they shall each and individually pursue, space is that moment of radical coevalness and contemporaneity (Massey, 2005; Lefebvre, 1991). Virtual space, therefore, is composed in the same manner as the apparently banal spaces of everyday life: both are the ongoing production of interaction. In this I am insisting that virtual space is indeed real space and is indeed *fully continuous with* real space, not by attempting to reduce the virtual to the apparent dullness of the spaces of the everyday but, rather, by attempting to elevate these 'real' spaces to the same level of novelty, to insist that they too are produced, are the outcome of process, and are constantly in the process of deformation.

Virtual space has traditionally been attributed some rather fantastic properties. In making virtual space fully continuous with other spaces, however, we must address and temper some of these claims. Namely, the speed at which information moves from point to point has prompted some to claim that the virtual has reduced space to a single point in which there exists only pure temporality. Paul Virilio, for example, has argued that cyberspace is a means of entering a world of immediacy,

> [where] having attained this absolute speed, we face the prospect in the twenty-first century of the invention of a perspective based on real time, replacing the spatial perspective.
>
> (Virilio, cited in Crampton, 2003: 10)

Claims such as these bear little relation to the lived reality of interacting with computer networks. One 'goes to' (or, alternatively, has a website visit them) only a few websites at any moment. Whilst interaction with a multitude of virtual objects is possible at any one moment, moving between these objects takes time, updating them requires refreshes and polling, and such interactions are always bounded by network latency which tends to be directly correlated with the physical distance of cables. This is not the collapse of space. Moreover, the claim of immediacy also fails to account for the network topology, such as the areas of a network that are forbidden or which are limited to a few by way of authorisation. To consider virtual space as collapsing in on itself is to consider the virtual only in the most abstract of terms, and fails to account for its heterogeneity and its concrete texture.[5]

---

5. Manuel Castells, in an inverse operation, has argued that we have created a 'culture of virtuality' in which we find 'the superse[ssion] of places and the annihilation of time by the space of flows and timeless time' (Castells, 2000:

To speak of virtual space as heterogeneous and as having a texture departs from the more common terms which speak of 'flows,' of 'connectivity,' of 'pure information,' metaphors which couple well with an idea of virtual space as both smooth and homogenous. Objects are absent in these descriptions, having apparently melted into air. In this study, however, I want to capture some of these flows, to talk about them in much the same way as we talk about spaces elsewhere, that is, to treat them as objects. Let us recall that when we speak of 'objects' we do not mean mere 'things' both inert and static, we mean processes that have produced *relative stabilities*, we mean machines which have come to produce an identity, which have delineated themselves from those things around them and which persist over time even as they may change. That is, this is a deliberate attempt at reifying those parts of the virtual that have achieved the stability that would otherwise grant them the status of object elsewhere. As part of this study, these objects will form some of the elements of the subsequent machines and enter into these accounts as full-fledged mediators. It is these objects of the virtual, in their interaction with one another and with humans, that give rise to its heterogeneous topology, that erect barriers, shape movement and interaction, and give the virtual its texture.

Having now established that virtual space is a part of real space, that it is, like the physical world, populated by objects, and that all these things are fully material processes, I want to 'reign in' one final aspect: the code. There is a temptation to conflate the code with a type of knowledge, and a temptation further to

381). This also seems to treat 'network' and 'virtuality' primarily as metaphors, paying little heed to their concrete form.

construct knowledge as a kind of free floating entity, existing everywhere and nowhere in particular. Moreover, labour that involves this free floating knowledge becomes recast as somehow 'immaterial' or as a form of purely 'knowledge work.'[6] Both our ontology and our reformulation of virtual space, however, forbid such moves. The Compiz code exists in the CVS repository within the Freedesktop.org server. This codebase, far from free floating in a netherworld of pure knowledge, occupies a bounded and limited space on a server whose access is carefully controlled and regulated. Work upon this object is a fully material process involving the ongoing accrual of code, of statements and expressions, functions and files, all of which come to reside in *some place* at *some time*. The Compiz project is a collaborative effort working upon a shared codebase, manipulating and transforming it in much the same way as a builders go about the building of a large structure. It is precisely in these terms that we shall be discussing the work within the Compiz project.

§This is a model of the production of social order that occurs wholly immanent to itself, a kind of 'self-ordering' that emerges in the interaction of heterogeneous elements — bodies, objects, spaces — and which remains ordered only for the duration of that interaction. There is no netherworld of structure, nor

6. The concept of 'immaterial labour' comes from the Italian Marxist tradition of *Operaismo* whereby it incorporates types of labour traditionally not considered labour as such, and which can be classed as either 'informational,' as with the case of our programmers, or 'affective,' in that they produce desires, norms and tastes. In both cases there is a strong service component to the work. See Lazzarto (1996).

an interiority of essence from which to derive this order, only a happenstance of elements against an inevitable backdrop of cosmological decay. When order does indeed emerge it is something for which we need to provide an account, to come to understand the machines in operation, their relations, and their effects.

In each of the three subsequent chapters we shall be exploring the elements of the three principal machines in operation both within Compiz, and of which Compiz is a part. It is in the ongoing movement of these machines, and their interaction and confrontation with one another, that gives Compiz its peculiar character. Each of these chapters begins by describing the operation of the archetypal instantiation of the abstract machine that forms the topic of the chapter, in much the same way as Foucault's Panopticon is the 'ideal form' of closed circuit television cameras. In starting with the ideal form, the task of drawing out the principal abstract relations of each machine is made somewhat easier. This is also to emphasise two additional features of this model: that abstract machines enjoy a mechanism independence where they may be instantiated within wholly different environments, and that the 'virtual' world which is of study here is not a special case, that it is not incommensurable to the spaces of the physical.[7] Indeed, it is precisely this commensurability that allows us to consider these machines of order apart from their virtual instantiation within Compiz and to consider them as utopian candidates for ordering social relations amongst wholly different elements to those found here.

---

7. I should also stress that the use of these archetypal machines is not to undertake a genealogy. Except perhaps in the instance of the last machine, there are no genealogical links, no *enstehung* (emergence) of forces that can be traced from the archetypal machine to the one at hand.

III. The Passport

Wherein virtual space becomes segmented and we discover the embrace of the user-space machine

The Passport is above all a spatialising machine. It is a machine that takes as its task the battle against anonymity. It seeks to produce identities that uniquely link bodies and objects with the interiority of its own documentary system, and to these identities are applied controls and monitors upon movement so as to effect a particular and desired distribution of these bodies across space. It is a machine that creates a striated and segmented space, a machine that we can class as both territorialising and hierarchicalising. But as it battles against anonymity, there are opposing forces that seek to escape its embrace, that transcend its borders or undermine its techniques. Acknowledged or not, the Passport machine is a permanent battleground. Within Compiz, the abstract machine of the Passport is instantiated as the 'user-space' machine, as the system of usernames and passwords, their associated permissions tables, and an array of technologies that seek to create boundaries. Through its production of space, it becomes the principal generator of hierarchy within the group. In this chapter we shall develop an understanding of the diagram of the

Passport, that is, its abstract relations by an examination of its historical development, before exploring both its operation and effects within the Compiz project as the user-space machine.[8]

§ The development of the Passport system and the modernist project of the nation-state were inextricably linked, for the state had as its subjects the nation, but the nation had no existence outside of the State. The nation could not be read off the body like other lines of demarcation — it was not defined by skin colour, language, cultural practices, religion or other traditional markers, though each of these could be indicators. It was an altogether modern and arbitrary division, one that could only come to be reliably elaborated through its progressive codification in documents and files. In his excellent study *The Invention of the Passport* (2000), John Torpey provides an important corrective to Benedict Anderson (1991), arguing that, 'in order to be implemented in practice, the notion of national communities must be codified in documents rather than merely "imagined"' (Torpey, 2000: 6). But if the development of the Passport was concerned with the identification and individualisation of national populations, it was also and immediately concerned with their regulation. The prevailing mercantilist policies throughout Europe from the 15th to the late 18th Century placed great emphasis upon the direct conversion of populations into wealth and military strength. To this end, the early Passport was deployed to aid in efforts of conscription, to control the movements of labourers

8. A note on usage: passport, lower case, denotes the actual document of the passport; Passport, capital letter, denotes the Passport machine as the total set of elements.

and especially skilled labourers, to tie serfs to their masters, to aid in the administration of poor relief and to control dangerous elements — gypsies, vagabonds, the wandering poor and, much later, the foreigner (Torpey, 2000: 18).

The implementation of passport controls at this time was thoroughly piecemeal and underwent a series of historical withdrawals. In absolutist Europe of the early modern era, travel of any kind was generally forbidden, except for those of the higher classes or else those in possession of a passport — although the ability for passport controls to be enforced was likely both poor and haphazard (Torpey, 2000: 22). For a period, the French revolution brought the necessity of passports into question with opponents arguing that they were a violation of basic human freedoms whilst advocates argued they were made necessary by the ongoing prospect of war (Torpey, 2000: 21–56). The revolutionary debates around the Passport were rendered null as France came under the rule of Napoleon and passport controls returned in full. Fewer than 100 years later, however, passport mechanisms were relaxed and virtually eliminated across most of Europe, at least in part due to the economic liberalism that prevailed for much of the 19th Century. The prescription of passport controls, however, remained in law, with their use stipulated as justified only in times of war (Torpey, 2000: 92).

While the Passport entered into a remission at this time, the individual technologies used in its implementation were elsewhere refined and elaborated. Each of its concrete components were deployed as part of machines elsewhere, such that the ongoing development of these machines fed into the advancement of the Passport machine itself. These developments included

bureaucratic techniques of managing files, knowledges about national populations derived from emerging techniques of censuses, the progressive delineation of borders, the systematisation of national identity documents, the creation of national police forces, and the development of 'anthropometric' techniques of identification in fingerprinting and photography, all of which would later converge within the Passport machine. There is a great historical contingency that must be stressed in the development of these technologies: each developed under their own logics, coalescing within various machines at different times, progressively elaborated and pulled by discordant forces.

At the onset of the First World War existing passport laws were reactivated, ostensibly as a temporary measure, and the different technologies that had been separately developing were brought together within a single machine. Passport controls during this time began to shift from being principally concerned with emigration to immigration, in part due to the abandonment of Mercantilist attitudes towards populations almost a century earlier and the concomitant desire to *shape* rather than merely *grow* populations. In spite of the cessation of war, passport controls remained in effect throughout most of Europe and North America, and were still in effect at the onset of the Second World War. War, once again, justified an intensification of the identification and control of both national and foreign populations, and the Second World War in particular proved a staging ground for optimising the different technologies used in the Passport. It was deployed both in the identification and control of those deemed foreigners, and in the control of the movements of 'internal' populations, the most infamous being the documentary and identification

techniques brought to bear upon the unwanted elements of the Third Reich (Torpey, 2000: 131–142).

In the period following the Second World War, the Passport machine underwent a kind of stabilisation, with the standardisation both of passport formats and the regulations between States, as well as the continued elaboration of the different technologies used. These technologies included the digital encoding both of passport information and of the correlative files, marked increases in border surveillance and enforcement, and new forms of anthropometric information offered by 'biometrics' (Jain, 2007). Even as the Passport machine became increasingly inviolable, there was a cautious relaxing of the control of movement across borders in some areas, notably within the European Union. In this case, however, the embrace of the Passport has remained in full effect, oriented primarily toward identification without which the various nationalisms — and the associated 'rights and obligations' of citizenship — could not be established (Torpey, 2000: 155).

§In this brief background to the Passport we can begin to draw out some of the elements concerning its operation. I want to construct from this history an 'ideal type' of the passport: the Passport as an abstract machine. I admit at the start a problem with this method, in that it assumes a sort of end-of-history conception of the Passport machine. Let me qualify, then, that it is altogether possible that there may arise future elements that should be included within this abstract machine. These potential new elements, however, will not simply be the result of technological innovation, just as the use of biometric data in place of the photograph does not change the basic relations of the Pass-

port. As I have stressed previously, an abstract machine obtains a certain degree of mechanism independence; what matters are not the concrete mechanisms but rather the relations they establish. We can ask, then, what are these relations? The Passport as machine is a number of elements of which the passport is just a part, a machine whose aim is the construction of unique and durable identities, the control of movement and the concomitant distribution of these identities throughout space. There are four main elements, namely, a set of borders, ports which function as surveilled places of passage, the files which track the identity within the bureaucratic interiority, and the document of the passport itself.

The segmentation of space is the first necessary condition of its operation, an altogether difficult task. Borders serve as an attempt to separate spaces, ports as the sanctioned points of passage between. In a flat space, borders may consist of walls, fences, tracts of water, and may exist under the active purview of bodies or objects — guards, cameras, alarms. Borders are often a violent and arbitrary separation of space, and their attempt to segment previously continuous space is often met with resistance. The border is always already a site of contestation: everything in its construction hints at an imagined force — the latent war machine, the nomads of the Steppes — seeking to transgress those objects and those bodies charged with its maintenance. And spaces change. Tracts of sea become traversable by ships, flat space is deformed into a third dimension with flight. Previously separated spaces become contiguous and the creation of new borders becomes necessary. But the border alone is confinement. It is the addition of the port that animates the Passport machine. The border functions not simply to disrupt flows, but

to redirect them through the system of ports. In contrast to the vast frontiers of borders, the ports are the points of concentration: they are manageable spaces of surveillance, identification, and flow. In contrast to the purely repressive effect of the border, the port is an apparatus of capture, a site of total envelopment, a bottleneck in which the asymmetry of forces reaches its zenith. In the port, bodies are subject to all the techniques of discipline: they are brought under surveillance, their movements are broken down, their bodies channelled through an apparatus of verification, with punitive and corrective mechanisms lying in wait. In the Passport machine, the port thus forms the site of modulation in the control of movement.

The diagram of the lock and key relies on little more than the border and a most rudimentary form of the port. The key, as a means of access, provides no means of identification, no means of logging, it does not aid in the generation of knowledges of populations, or in the embrace and shaping of those populations. It is everything beyond the border and the port that makes the Passport proper. Of these additional elements perhaps the most important is the attempt to create a durable and unique correlation between the body or object and the bureaucratic interiority: Weber's 'files' (Weber, 2004: 246). Somewhat in contrast to the battleground of the border and the port, the banal activity of writing and record-keeping contained within the files is foundational to the Passport. Foucault puts this well when he proclaims that the examination,

> […] places individuals within a field of writing; it engages them in a whole mass of documents that capture

> and fix them. A 'power of writing' was constituted as an essential part in the mechanisms of discipline.
>
> (Foucault, 1995: 189).

The files — the catalogue, the registry, the database, the log — create a correlative identity to the body, one composed primarily of writing. Its ideal is perfect correspondence: what happens to one happens to the other. The files may be prescriptive: they may describe the restrictions and allowances that are to be imposed upon the body or object. They may be historical: providing a log of movements, interactions, and histories. And they may be descriptive: descriptive of the body, its characteristics, its manners.

But the correlation between the body and the files requires a link: the pineal gland of the Passport machine is the passport itself. To the passport is designated the work of establishing this link, maintaining the unique and durable correlation, pointing at once in both directions. To the first direction, towards the body or object, it became necessary to uniquely identify the body. This has at times been achieved by practices of writing on the body — branding, tattoos, bracelets and armbands — but as these techniques have become relegated primarily to the animal, the subhuman, and the prisoner, alternative techniques were required not to write on, but to read off of the body (Torpey, 2000: 17). At first, this was merely a description of appearance such as eye colour and hair, height and sex, but it was progressively enriched with other forms of anthropometric identification: signatures, the development of photography, fingerprinting, and more recently the use of biometric data such as facial and iris recognition often encrypted on chips embedded within the pages of the passport. In the opposite direction, towards the files, the correlation is easier to establish thanks to the techniques of indexing and cataloguing:

the mere mention of the file number. In all these efforts there is the need to guarantee the irreproducibility of the document of the passport. Early technologies included elaborate ink patterns and sophisticated printing techniques, watermarks, lamination, barcodes and more recently the application of digital cryptography. We sense once again a kind of battle: there is an allusion in these efforts to opposing forces that seek to disrupt the correlation that the passport attempts to create between the body and the files, that each of these moves seeks to more firmly grasp the body and bypass those efforts that would circumvent this embrace, to guarantee in the passport truth.

The Passport is a machine that is both repressive and productive. It is repressive insofar as it denies movement, insofar as it segments space, where it may be deployed to limit or deny access to all manner of practices and organisations. But it is also productive. It is used not just to deny people access to spaces but to construct spaces of a special kind, to *embrace* whole populations within administrative apparatuses so as to shape them, act upon them, to come to know them, to make whole populations useful, and to surveil them. This is, for John Torpey, a critical point, and we can see in our previous quotation from Foucault a similar insistence: to 'capture and fix.' Torpey contrasts this notion of embrace with the view of the State as penetrative, as standing outside of populations and applying its machinations of power at a distance. But, he insists, the State must first embrace populations in order to penetrate them: 'the reach of the State, in other words, cannot exceed its grasp' (Torpey, 2000: 11). In bringing individuals into a realm of documentary controls and identification the State brings the social body into itself.

§ Weber wrote of the State as the expropriation of the means of violence, Marx that capitalism was the expropriation of the means of production and, Torpey argues, that the Passport is the expropriation of the legitimate means of movement. After an era in which non-State entities had considerable control over the means of movement — the Church, feudal lords, certain members of the aristocracy — Torpey argues that these abilities have been subsumed by the State, with private entities reduced to the capacity of 'sheriff's deputies' (Torpey, 2000: 9). It is here, however, that Torpey unnecessarily limits his analysis. By insisting upon the exclusivity of the Passport to the domain of the State, the diffusion elsewhere of the Passport as abstract machine is lost. That is, by insisting upon the Passport machine as the sole prerogative of the State we lose the ability to identify instantiations of the Passport elsewhere, to perceive how the Passport has spread bit by bit and been enacted within progressively smaller spaces by all manner of non-State entities, enacted in ways that are not usefully reduced to a mere 'deputy' status of the State. Schools, factories, workplaces, universities, apartment buildings, gated communities, public transport and even city centres have all been sites in which the Passport has been enacted, in which the four elements of the Passport machine have been brought into proper relation to one another. School identification cards that monitor truancy, passcards that enable and track access to buildings and the spaces within, or transport registration systems that are deployed to monitor access to city centres so as to impose pecuniary costs to the correlative bodies: each of these instantiates the Passport machine towards the different ends of security, surveillance, the management of space, or otherwise.

The virtual is a realm that has since its inception progressively enfolded the Passport machine within its standard operation, purportedly in the service of 'security.' This stands in marked contrast to some popular accounts that ascribe to virtual space a kind of smooth and unbounded quality, a horizontal and flat web of networks that are democratic, free and egalitarian by nature.[9] In this formulation the virtual stands in opposition to all the barriers that inhibit movement in physical spaces, as a kind of emancipatory realm of pure connectivity. But, while the network itself is both flat and decentred, the nature of those elements connected within the network is far from emancipatory: each node is a fiefdom unto itself, a fortress strictly determining its routes of access, strictly dominating those elements to whom they have granted access, and strictly shaping the manner in which that access is granted. To those who have undertaken the task of its reproduction, the virtual is conceptualised simultaneously as a domain of possibility and threat: the crafting of each of its possibilities involves the careful consideration of potential risks, 'attack vectors,' software bugs, and the implementation of carefully bounded security models (for example, Arctec Group, 2005; Fernández-Medina, et. al., 2006). It is a model that seeks to control and identify the masses of anonymous elements, and

9. Nicholas Negroponte (1998), for example, writes that that social inequality is an 'artifact of the world of atoms,' not cyberspace. Louis Rossetto, founder of Wired Magazine, claimed 'This new world [of the Net] is characterized by a new global economy that is inherently anti-hierarchical and decentralist, and that disrespects national boundaries or the control of politicians and bureaucrats […]' (cited in Barbrook, 1999). And in John Perry Barlow's famous 'Declaration of the Independence of Cyberspace' he writes, 'We are creating a world that all may enter without privilege […] Your legal concepts of property, expression, identity, movement, and context do not apply to us. They are all based on matter, and there is no matter here' (1996).

is characterised by a never-ending concern with patching newly discovered 'vulnerabilities' both in the software and in the widely-used encryption algorithms.

The maxim of the contemporary security model, popularised in the original Unix operating system, is the enclosure of every element of a computer system by way of the 'user-space' machine. The user-space machine operates at the very heart of most operating systems, wherein virtual objects such as users, programs, files, directories and devices come to be embraced by this model. The means through which this is done is by assigning to each object a user ID, which functions as the equivalent of the document of the passport.[10] The list of acceptable user IDs is kept in a set of system files which also record the user's password and a number of system wide settings. Meanwhile, every single object is also assigned a set of additional permissions which prescribe the actions any particular user ID can perform upon that object, such as whether a user can read the object, whether it can write to and edit the object, and whether it can execute the object as if it were a program. These permissions, which form the equivalent of the files, are mapped upon each element by way of the system's file system and enforced by the kernel at each attempt to traverse the port of a virtual object.[11] Files and whole directories can, in this manner, be restricted such that they can be seen but not touched, or such that they are off-limits and hidden entirely. Each of these

10. Each element is also assigned a group ID, but this makes the discussion more technical than it needs to be.

11. Files and other objects are accessed by programs (and therefore users) by way of a 'system call' which asks the operating system's kernel to act upon a virtual object. In this way, the kernel becomes the obligatory point of passage and performs the function of the port.

virtual elements is thus brought into a regime of system permissions, whilst additionally all manner of actions are logged — from login attempts and access to system resources, to network traffic and manifold interactions with the security model.

The ideal operation of this system is the granting of just enough permissions to any particular user so that they may perform their task, whilst granting not a single permission more. In opposition to the notion of a smooth and frictionless space, this is a system of maximum enclosure, where each of these elements becomes bound to a restricted domain of the computer or network, unable to do anything it need not do, unable to touch anything it need not touch. The user-space machine is a security model characterised by a concern with a strict adherence to a particular spatialisation of the virtual: a fully-fledged Passport machine.

Within the aforementioned user-space machine operated a program known as CVS or the 'code versioning system' which monitored and controlled code entering and leaving the Compiz repository. This program typically operates under the aegis of a unique user on the computer, and thus was already confined to but a small portion of the computer's resources. Additionally, however, CVS implemented its own internal user-space machine, an implementation which we shall explore here by tracing the steps through which one would gain access to the code contained within.

To begin, then, on my own computer I start my CVS client which provides me with the means to interact with the CVS server across the Internet. From the Compiz wiki I have obtained a couple of commands with which to gain access:

```
$ cvs -d:pserver:anoncvs@cvs.freedesktop.
org:/cvs/xorg login
CVS password: <hit return>
$ cvs -d:pserver:anoncvs@cvs.freedesktop.
org:/cvs/xorg co app/Compiz
```

(Anon., 2006)

These commands say a number of things, two of which I want to note here. The address 'cvs.freedesktop.org' is the first thing we should note: it is much like any normal web address in that it gives us the ability to find our desired server amongst the thousands of others connected to the Internet. The second thing we should note is the username through which we gain access to this server: 'anoncvs.' As one might infer, this username is not unique to myself but is instead provided to allow a collective hoard of unknown and anonymous users access to the code. By operating under the aegis of this username I am, on the one hand, enabled to view and read the code, and to download it to my local machine. It is precisely through this anonymous access to the CVS repository that Compiz becomes 'open source.' On the other hand, by operating under this username I find myself immediately embraced: I am pulled into the regime of permissions of anonymous users and I can be tracked in my movements, though this embrace is limited so long as this username is used by a group.

When I enter the command a number of things happen. My computer sends out requests across the Internet, marked with the desired server address as its destination. My request eventually reaches its destination, at which point it confronts the network equivalent of the border and the port. This border is a software barrier at which I can fire all sorts of network requests that should dissipate to nothing. It is not so much a physical entity as it is a

process: it exists in the way it handles the constant barrage of network requests, it is performed through these interactions of rebuff and diversion, and when it fails in these roles it ceases in that moment to be a border. This border is made up of thousands of ports, each potentially being 'listened' to by a program on the remote server. My CVS program knows ahead of time, as part of an already-established protocol, the default port it should attempt to use to gain access: CVS will be listening on port number 2401. This port is the site of my entry into the CVS component of the server; it is the single point through which my network requests are channelled, verified and monitored. The port is also the point of capture. My CVS client sends a request to this port, '**`BEGIN AUTH REQUEST`**,' followed by my username 'anoncvs,' my password, which in this case is absent, and a final closing string, '**`END AUTH REQUEST`**.' To this I wait for a response. If I receive the response '**`I LOVE YOU`**' my CVS client will proceed to negotiate my entry into the repository, or else '**`I HATE YOU`**' informs me that I am denied access (Anon., 2000).

How does the server know to let me in? And once I have entered, what can I do? This introduces us to the virtual equivalent of Passport's files. Contained on the server are three files. The first, the 'passwd' file, establishes the usernames and passwords that the CVS server will recognise. This is nothing but a simple text file, with each line in the form of,

```
CVS_username:password:system_username
```

The file establishes a correlation between each of these elements: my CVS username and my password are linked together, and these are then linked to an internal server username which, by references to the system permissions, regulates my access to the

various system resources. There is in this simple file an element of the tripartite system of identification of the Passport machine: the body/CVS_username is linked to the bureaucratic files/system_username by way of the intermediate device that is the passport/password. There are two more files, equally as simple in their format. One, a readers file, lists those users with 'read-only' access to the repository: they may view the code contained within the CVS repository, they may copy the code to their own systems, but they may not edit or change the remote repository. The writers file enables write access to the repository. Usernames listed here may both read and write to the repository: they may make 'commits' to the codebase and edit it as they desire. These reading and writing restrictions are enforced by the CVS program running on the server. Moreover, the CVS system must ensure that the reading and writing of files occurs only in limited and designated directories. As per our user-space regime of maximum enclosure, all areas beyond the code itself are strictly forbidden, the enforcement of which falls to a combination of the CVS program itself and the remote server's own internal user-space regime.

This is all rather complicated, but its effects are simple: as an anonymous user I can only read the code and copy it to my own machine. No more. And these actions — my logging in and my 'checking out' of the code — are all logged, recorded in large text files. Moreover, since the anonymous user was predominant, this was the typical experience within the Compiz project.

§For anonymous users within Compiz, to contribute patches back into the code repository it was made necessary to sub-

mit these by way of a mediator: David Reveman. David was a special sort of registered user known as a 'super user': not only could he both read and write to the repository, he could control all manner of finely tuned access controls. It was to him that the user-space machine granted the ability to craft permissions tables. The various people who contributed to the Compiz code thus had to first email their code to David Reveman who would 'review all patches before they [went] into the CVS' (Reveman, David: 2006-04-04 05:32). It was through the user-space regime of CVS, and through his unique access to its space, that David Reveman became elevated to the status of a special and necessary mediator: the gatekeeper. The immediate effect of this role was a unique and unparalleled control over the progressive shaping of the objects Compiz-as-code and Compiz-as-object. But it had another very important effect that extended beyond the object of the code. So long as contributors desired to contribute to Compiz-as-code, these users found themselves compelled to comply with both the coding practices and the project direction of the gatekeeper. By way of controlling the object Compiz-as-code, the user-space machine granted to David the ability to exercise a limited kind of power over these willing contributors. This kind of power we shall label as 'vicarious.'

We saw in Chapter One some of the gatekeeping manoeuvres with respect to patches that were executed by David. These can be classed into three categories. The first was an unhindered commit, whereby a patch was submitted to David who proceeded to commit the patch into the repository unedited or, alternatively, advised the submitter to commit the patch directly themselves. The second was a modified commit, whereby either David accepted a patch, edited it himself and proceeded to commit, or

alternatively advised the submitter on certain changes that were required before he would accept and commit the patch. The third was a rejected commit, where the patch was altogether denied being commit to the code repository. Let us take some examples of these triaging practices.

Patches passing into Compiz unhindered were relatively rare, with most being modified in some manner. The majority of these patches were submitted to the email list, and thereafter committed by David into the repository. On 18 April, for example, Gandalfn submitted a patch to the list which 'add[ed a] command line option to force bind and release whenever texture is used' (Gandalfn: 2006-04-18 12:30). Later that day, David announced that he had modified the patch and committed part of its functionality to the repository, but that he had rejected another portion of the code claiming that 'that's something that should be fixed in the server and not in compiz' (Reveman, David: 2006-04-18 17:51). Later on in the project, however, as individuals gained write access to the repository — a point we shall cover shortly — they were given consent to commit to the repository directly. On 20 September, for example, Kristian Høgsberg submitted six patches to the list, four of which David authorised for immediate commit (Høgsberg, Kristian: 2006-09-20 07:27). The other two underwent a modified commit: on one David advised that he intended to alter the patch after it was commit, and on the other he asked Kristian to make a number of edits (Reveman, David: 2006-09-20 08:36). Later that day, the code repository records that each of these patches was submitted directly by Kristian. In requesting Kristian to edit his code according to David's prescriptions, there is a subtle shift from David merely editing the incoming code to exercising a degree of power over those submitting the code.

The modification or rewriting of patches by David was the most common form of patch triage; the rejected commit was rare and controversial. In Chapter One we covered some of the controversy caused by the rejection of both the shadow colouring patch and the multi-screen 'Xinerama' support. To the first, David defended his rejection by claiming that colouring window shadows was a kind of 'useless crap' best kept out of Compiz as it only added to the amount of code requiring ongoing maintenance, and to the second, that Xinerama was an improper method to implement multi-screen support (Reveman, David: 2006-05-03 03:38; 2006-06-19 06:25). In both instances, numerous contributors questioned both the validity of David's decision — whether indeed they were really useless, or whether it was an improper method — as well as his right to make those decisions in the face of dissenting opinions (see Storm, Quinn: 2006-05-03 09:39; Szulecki, Martin: 2006-06-19 04:26).

These decisions around the triaging process compelled a number of people to ask David to provide explicit rules and guidelines for writing code. Guillaume Seguin, for example, wrote on 24 June,

> Most patches written by these developers get committed to Quinn Storm's cvs […]. Nevertheless, it seems that unfortunately little of the very good work that is done actually gets into the official project, which can make it difficult to continue to be enthusiastic about developing for Compiz.
>
> As a group we were wondering what the standards that any plugin or patch must meet before it can be included in the main codebase are: functions/constants/variables naming convention, coding style… Moreover,

> if there is some reason that patches are not applied […] is it possible to share it with the list […].
>
> (Seguin, Guillaume: 2006-06-24 16:24)

Thomas Liebetraut wrote to the list the next day in agreement with Guillaume,

> That's why I would appreciate informations [sic] about the patch standards, too, because it's somehow frustrating to know that the work you did during the last weeks will end up in the trash can and someone else rewrites your patch from scratch.
>
> (Liebetraut, Thomas: 2006-06-25 06:11)

Thomas' comments allude to another power effect of the triage process. Not only could David ask contributors to edit their contributions — a quite explicit exercise of power — but it appears that his editing and rewriting of patches compelled some developers to anticipate David's desired standards in an effort to avoid seeing their work come to naught. To these requests there was no reply. In an exchange considerably later on 15 September, Shawn Starr wrote,

> I would like to get a better understanding as to why Quinn's patches have not been accepted into the compiz git tree.
>
> I feel Quinn's patches could greatly improve compiz. She has made quite a lot of progress [with regards to] enhancing compiz with her cgwd window decorator which allows users to write their own themes for window decorations.
>
> (Starr, Shawn: 2006-09-15 12:03)

To this David replied,

> I am definitely willing to accept patches but I won't push in some ugly patch just because it adds some additional functionality, I [would] rather wait for the proper solution (Reveman, David: 2006-09-15 13:13).

His apparent reluctance to provide anything more explicit than avoiding 'ugly' code, and implementing 'proper' solutions echoed previous exchanges and was not well received for it was shortly after this email that the Compiz/Beryl fork occurred. On 28 September, David spoke on the topic of the fork confirming that he believed that,

> With a few notable exceptions, most of the code I've seen going into Beryl is not high quality code that would be considered for Compiz.
>
> (Reveman, David: 2006-09-28 10:12).

Two weeks after the fork, David provided stylistic but not technical rules for code in Compiz in which he documented the conventions around tabs, function and variable names, code width and a number of alignment conventions — some of the guidelines for which Guillaume had asked four months earlier. He concluded, 'looking at the code is the easiest way to get what coding style is used' (Reveman, David: 2009-10-05 13:12).

The gatekeeping role that enabled David to triage and modify code enabled him to uniquely determine the unfolding of Compiz: from the choice of code styling conventions, to the designation of certain features as useful or not, from the determination of poor coding to key decisions around code architecture. Each of these aspects of Compiz was, in the last instance, the prerogative of the repository gatekeeper, a role that was an effect of a largely unstated but immediately felt spatialisation enacted by the user-space machine. In each of these instances, it was this spa-

tialisation that produced the obligatory point of passage of the gatekeeper. This was a type of power, no doubt, but it was an odd form of power. It was not the direct control over other people with which we are more familiar. David could not order anyone directly. Instead, this was a tenuous 'power-over' vicariously enacted through David's control over an object. It was through other's desire to work upon Compiz that the gatekeeper role was extended beyond direct control over Compiz-as-code into a weak form of power-over: the power to shape people's contributions, to encourage or discourage areas of work, to regulate their participation. Moreover, this power was binding only insofar as the object itself — the code — remained designated official, that it remained the exclusive avenue for work to proceed, and only so long as others desired to contribute to the project. It was therefore a power effect both vicarious and precarious.

§This was an important though relatively simple effect of the Passport machine: the control over the flow into and out of the code repository, where that code was monitored but free to travel in one direction, and directed through a single discretionary actor in the other. It was also a rather stunted implementation of the Passport. The ability to fully individualise a population was lost: access to the code repository was granted through the collation of many and different people under a single username, and the submission of code occurred via email, largely escaping the embrace of the Passport machine altogether. Moreover, this implementation was primarily repressive in its effects: it operated to restrict the flow of code into the repository but, besides the creation of the gatekeeper role and its concomitant ability to shape

the code, it had few other productive effects, either in the creation of spaces or in the moulding of populations. This is to say, both the productive capacities of the Passport and its individualised embrace were not fully realised in this anonoymous model.

Early on in the project, however, at the same time as David announced his intention to review all patches before being commit into the repository, he also announced that,

> If you have got a plugin or decorator, I'm more than happy to put it in CVS and give you commit access as long as you're willing to maintain it, there's a configure option to disable it, and it's not a complete piece of crap.
>
> (Reveman, David: 2006-04-04 05:32).

The plugin and the decorator rely on a principle fundamental to programming and computer systems known as modularisation. This abstract machine will form the basis of Chapter Five, but for now we need a basic understanding of the concept. Modularisation is a process of 'black-boxing,' of hiding away complexity behind a simple and, more importantly, a stable interface. A program without modularisation is a highly interdependent meshwork of code: each function may rely on another function elsewhere, each variable may rely on being set by a diverse number of routines. As a program gets larger, the complexity of these relationships becomes ever greater and the changing of one part of a program may have great and potentially disastrous effects elsewhere. Modularisation attempts to resolve this problem of complexity through the spatialisation of code. It involves the segmentation of code into a limited number of bounded spaces, the interiors of which are designated as off-limits to code beyond, coupled with the creation of externally accessible façades, known

as interfaces, to handle communication between each modular component.

Compiz-as-code had its own high-level modularisation, split between 'core' and a large number of plugins. Compiz core contained the code that had been deemed by David Reveman as essential code, code that he believed was of a universal nature and should be made accessible to all other parts of the program. In addition to providing these universal functions, Compiz core also implemented a plugin architecture: it created a set of routines, interfaces and protocols that could allow a self-contained plugin to be inserted into the code. These plugins added new capabilities to the Compiz core skeleton, which by itself did very little: one added shadowing to windows, another made windows wobble when they were moved, and another moulded the virtual desktops onto a cube-transform. Just as each could be added and removed from the code cleanly thanks to this modularisation, so each could be disabled or enabled within the Compiz program.

Upon this modularisation of code was overlaid the user-space machine. Just as modularisation segments the code, transforming it from a dense mesh of interlinking code into functionally separate spaces, so too the programmers may be each assigned a domain, transformed from the horde into a more individualised state. Each of these spaces came under the controls of the user-space machine, permissions tables were drawn up to carefully regulate users' access to the different spaces of the code, and logs tracked minutiae movements across the newly constituted borders. Given access to a modular space within the code repository, a programmer could be *enabled* to write code, code that could access Compiz core via the plugin interface, code over which the

programmer could be granted both read and write permissions and therefore bypass any form of direct gatekeeping. In the same instance, the programmer would be brought within a space of enclosure, limited to the domain of the module, to the functionality granted to plugins, and denied the same write permissions to other plugins or core.

This *enclosure* forms the first component to Foucault's 'art of distributions,' which was for him a type of disciplinary individualisation (Foucault, 1995: 167). He described three further aspects to this art, each of which we find fully articulated within the combination of the user-space machine and the modular framework of Compiz. The enclosure of each programmer within their own modular space had a concomitant effect: that the population of programmers found itself broken apart, each designated a space of their own. For Foucault, this *partitioning* was a method of managing the perennial problem of the horde or the mass:

> Each individual has his [sic] own place; and each place its individual. Avoid distributions in groups; break up collective dispositions […]. Disciplinary space tends to be divided up into as many sections as there are bodies or elements to be distributed.
>
> (Foucault, 1995: 143).

To this second aspect of the art of distributions was added a third: *functional sites*. With the allocation of each space came the allocation of a task. The creation of code within each of the modular spaces of Compiz was limited by the affordances offered by the plugin interface, which is to say, by dint of their architecture, each of these modular spaces became functionally oriented towards writing plugins, and only plugins. This was both a repressive function of power, limiting activities within these domains to a desig-

nated subset, as well as a productive function, for it simultaneously operated to create these spaces as useful, as equipped with the necessary tools and interfaces for plugin development (Foucault, 1995: 144). The *table* was the fourth aspect of Foucault's art of distributions, and operated to classify bodies or objects according to a spatial register. The hierarchy of programmers within Compiz was overlaid simultaneously onto a spatial distribution of sites: David Reveman operated within the universal space of core; the limited number of plugin maintainers presided over the subspaces of the plugin modules; and at the bottom, the anonymous contributors operated within the no-space of mere access. This particular disciplinary individualisation — employing enclosure, partitioning, functional sites, and the table — Foucault termed 'cellular' (Foucault, 1995: 167).

The cellular individuation offered by the modular spatialisation of Compiz coupled well with the gatekeeper role, and functioned as a kind of stopgap. In the first instance, it allowed a more active participation within Compiz according to the affordances offered by the plugin interfaces and relieved coders of much of the frustration of the triage process. This had the effect of reducing the dissatisfaction concerning the spatialisation of Compiz and its effect in the gatekeeper role. But in the second instance, it was operated as a containment strategy, enclosing and partitioning coders and their code, both of which could be trivially disconnected from the Compiz project owing to their careful boundedness. This twofold process of affordances coupled with containment is made quite explicit in a number of exchanges on the email list. In an early exchange at the end of June, the possibility of forking was being raised by several contributors prompting the response by Matthias Hopf that,

> I don't really think [that there is the need for divergence]. As compiz is composed of plugins, alternative plugins should be possible. So if the goals for a particular plugin are too different to be solved in a single source fragment, only the particular plugin should be forked (inside the same repository).
>
> (Hopf, Matthias: 2006-06-26 03:35)

Matthias' emphasis is on the containment of dissent, consigned not to Compiz as a whole but to the limited domains of the plugins, and he stresses the overall unity of the project even as individual plugins may be forked. Following the Compiz/Beryl fork, David wrote to the list on 28 September arguing that the reasons for the fork were ill founded as he had,

> […] designed compiz to be extremely extensible. The plugin system should allow people to do almost anything and I've put a high priority in making sure it got updated when I or someone else found something that couldn't be done with it. People can choose whatever development methods they want and put whatever code they want into plugins.
>
> (Reveman, David: 2006-09-28 10:12).

There is once again an explicit appreciation of the role that the subspaces of the plugins were meant to play in heading off the possibility of the fork.

§The means through which the user-space machine was enforced within the spaces of the plugins is testament to its unique logging abilities. For elsewhere the user-space machine typically operated according to a system of simple control mechanisms: what was allowed was made possible, and that which was

not allowed, rendered impossible. There was in this no possibility of deviance. But in the operation of the plugin spaces, the usual mechanisms deployed to enforce maximum enclosure were not implemented. Instead, there was a departure from a regime of control to a regime of norms, made possible by the enaction of a condition of transparent traceability. Here, the Passport machine's ability to catalogue movements was deployed such that deviance, now possible, was also and immediately made visible.

The technical implementation of the user-space machine within the Compiz plugins was implemented through the limited allocation of individual usernames to individual plugin maintainers. In place of the anonymous and collective username 'anoncvs,' one chose instead a unique username. But for this limited few, the greater permissions that were offered by individual usernames came at the cost of a significantly firmer embrace by the user-space machine. This embrace required a firmer correlation between the body and the username and it fell upon the object of the password to establish this link. The password had two effects. The first was to keep undesired users from accessing the repository through other's accounts. But it was, in addition, an attempt to firmly and uniquely link the body and the username. The password, as a tidbit of information that ideally existed in the mind of just one person, served in this context as a kind of anthropometric identification. Moreover, its integrity was guarded through the use of advanced techniques of encryption. In this manner, the password was the guarantor that each username uniquely correlated with the body to which it was assigned.

It was this embrace, now firmly established with the individual usernames, the password, and techniques of encryption, that

could generate a condition of transparent traceability. Much as the files of the Passport machine are both prescriptive and historical, so too the user-space machine contained both permissions tables as well as activity logs. Moreover, these activity logs were of a much finer precision than the Passport machine of States, for almost every object and every space was a border, and thus every kind of activity prompted some sort of interaction with the user-space regime. For every kind of activity that occurred within the CVS repository, an entry into the record log was made, recording the activity, the time, a number of more esoteric datum and, most importantly, the username. Every commit made was immediately connected to the username and the body that had made the commit, each edit or reversion was logged against the unique identity that performed the operation, every traversal of a virtual border was recorded. This condition of traceability was not only total but it was, for the most part, transparent: other users, both individual and anonymous, could view these records.

It is in this condition of transparent traceability that we can fully appreciate the nature of the departure from the more common regime of control of the user-space machine to the limited space of norms within the CVS repository. Those given plugin access could in fact make commits beyond their modular spaces: the possibility existed for them to commit into core, for them to edit other users' plugin code. That is, for them to be deviant. As Matthias Hopf wrote at one point on the email list,

> Getting a CVS account for X.org is not complicated. However, you should only commit yourself into a particular tree, if the maintainer agrees. This is basically David's decision.
>
> (Hopf, Matthias: 2006-06-26 03:35).

The operative word here is 'should,' signalling the possibility that users could do otherwise, and as Matthias indicates, it was David's ongoing role as super user that bestowed upon him the ability to define behaviour as proper or improper. Deviations from his prescriptions would have been immediately made visible and would have been uniquely linked to the body by way of the user-space machine. Moreover, the CVS repository guaranteed that any such improper commits could be reversed to an earlier version, thus rendering the repercussions of deviant behaviour as trivial. With the exception of those instances where David gave explicit license to registered users to commit particular pieces of code to core, there was not a single instance of deviant behaviour by a registered user during the period of this study.

§Virtual space and, in particular, the space of Compiz is neither smooth nor continuous. It is a highly segmented space, carefully managed by its own instantiation of the abstract machine of the Passport. These key elements — the border, the port, the files, the passport, and the host of other technologies used to stabilise each of these elements — could be found in the user-space machine of the Compiz code repository. This user-space regime produced two different kinds of effects. The first concerned the anonymous user, whereby the user-space permissions allowed for an unrestricted flow of code out of the repository but denied the ability to commit code in the other direction. This twofold process of spatialisation and access controls enacted the system 'super user' as a kind of gatekeeper, one through whom incoming code had to first pass and who, by dint of this fact, gained the unique ability to shape the progressive development

of the objects Compiz-as-code and Compiz-as-program. In this ability, David Reveman as gatekeeper came to exercise a kind of vicarious power throughout the course of this study. The second kind of effect concerned the production of the modular spaces of the plugins and the creation of the registered user. To this user was granted not only read access to the CVS repository but write access too, with the expectation that write access would be restricted to their designated plugin space. This was a process of affordances coupled with containment, and was intended to function as a stopgap to the growing dissatisfaction caused by the gatekeeper role and its sometimes unpredictable triaging of patches. To the registered user the additional permissions came with a much firmer embrace by the user-space regime, with the password and the computer logs combining to enact a condition of transparent traceability that ensured compliance with the prescriptions of the CVS super user.

If the Passport effects a spatialisation whose effects are primarily territorialising and hierarchicalising, then the abstract machine to which we next turn stands at odds on both accounts. The Fork, similarly built upon its own kind of spatialisation, is deterritorialising and anti-hierarchical. The existence of space elsewhere outside of Compiz coupled with the open source property regime brings to the fore a 'precarious' aspect to the gatekeeper role, and it was in their juxtaposition to one another that the Passport and the Fork produced one of the major tensions within the Compiz project.

IV. The Exodus

In which talk of escape renders the powerless as powerful, and the gatekeeper is transformed to maintainer

The abstract machine of the Exodus operates as a kind of sword of Damocles, perpetually raising the possibility of desertion through the construction of an 'imaginary counter-power.' In this it confronts an order with an ongoing spectre of division, one that in its threat of resistance comes to wield constitutive effects upon the nature of that order. The abstract machine of the Exodus tends to undermine the centralised exercise of power and thus comes to be decentralising in its effects. Within the Compiz project, this diagram was concretely instantiated as the 'forking' machine owing to realisation of two conditions. The first condition was in making the prospect of the fork a genuine possibility by the provision of both sufficient resources and of space elsewhere. Secondly, the possibility of the fork was made known by a discourse of counter-power, one that constantly reiterated the prospect of the fork. With these two conditions fulfilled, the project operated under the constant threat of a desertion and the looming prospect of division. The primary effect of the forking machine was to impose an additional role upon the gatekeeper

known as the 'maintainer,' a role that recast the gatekeeper in service to the community of developers and tasked him with the ongoing maintenance of the code. It was a machine, therefore, that countered the power of the role of the gatekeeper by locating within the community of developers an additional and opposing locus of power.

§To understand the machine of the Exodus, we need to first understand the notion of both counter-power and its form in potentia known as imaginary counter-power. The notion of counter-power is an idea prevalent in anarchist and anti-State social movements. It is also known as anti-power and, in certain situations, as dual power.[12] Counter-power exists within a group as a plethora of machines — institutions, groups, material resources, media and so forth — that are opposed to and set against the hegemonic order of that group. It is an opposition to a dominant order that forms within the bounds of that order. But it is more than mere opposition. A conglomerate of machines only comes to resemble a counter-power when it contains within itself a substantial *constructive* moment: its own form of social organisation, its own ordering mechanisms, and its own provisions, however weak or tentative, for providing for its own needs. A counter-power opposes a dominant order through the enaction of its own form of social, material and spatial organisation. This alternative form of organisation is not a *replacement* of existing power, it does not seek to 'conquer and get [its] hands on the old power, but to

12. See John Holloway (2005) for an elaboration on 'anti-power.' 'Dual power' was first articulated by Vladimir Lenin (1917) in an article by the same name.

develop a new *Potenza* of life, organisation and production' (Negri, 2008: 144; emphasis in original). The opposition of counter-power is thus threefold, as resistance, insurrection and constituent power: 'insurrection pushes resistance to become innovation […] and, whereas the insurrection is a weapon that destroys the life-forms of the enemy, constituent power is the force that positively organises new schemes of life' (Negri, 2008: 140).

In its opposition counter-power comes to wield a constitutive effect upon the dominant order of the group. Some version of this idea has animated Marxism for well over a Century now in the form of the dialectic forces within society that drive it through the progressive stages of History. It finds its most explicit expression in the Italian Autonomia movement and its notion of the constitutive force of the multitude, no longer deemed the hapless and passive subjects to the movement of History. Antonio Negri, for example, writes that 'the state […] is organised to control and repress counter-power' and that 'the struggles as extreme and powerful danger are always present, obsessively pressing on the capitalist definition of development' (Negri, 2008: 145). The machines of hegemony are constantly propelled by both real and imagined threats to their dominance, always and already engaged in reconstituting themselves in opposition to the movements of counter-power. They are, that is, partially constituted by counter-power.

One step removed from the notion of counter-power is *imaginary* counter-power. Imaginary counter-power, unlike real counter-power, does not exist as a concrete and already-existing seed of an alternative social order. Rather, it exists as the *potential* for machines of counter-power to form, as the ongoing and latent

possibility of the emergence of real counter-power. The emergence of counter-power represents a declaration of open battle over the form of social organisation, whereas imaginary counter-power is the ongoing threat of such contestation. The elements of imaginary counter-power are different to its concrete cousin: they are those machines directly tied to the aspect of its potentiality, those elements that fulfil the necessary conditions for the emergence of real counter-power. In the ongoing potential for counter-power to form, and *to form with ease*, imaginary counter-power itself comes to wield constitutive effects upon a hegemonic order, effects that are directly proportionate to the perceived prospects of its appearance.

As an example, let us take from the work of the French anthropologist Pierre Clastres who documented a type of imaginary counter-power that operated amongst a number of tribes of the Amazon. Clastres wrote in opposition to an anthropology at the time that suggested that these societies, being without State apparatuses, were an infant form of political order, that they were in some way deficient in comparison to those societies with States. In presuming the natural evolution of societies to be one towards Statehood, the onus therefore fell upon Clastres' contemporaries to explain this lack. Clastres took a novel approach and instead proposed that these were societies *against* the State and suggested that their whole social organisation was oriented against the emergence of hierarchy, against the 'spectre of division' between the dominating and the dominated. For Clastres, these societies were no longer in some way deficient and incapable of progressing along the evolutionary path of political forms, but rather that possibility was always and already foreclosed by the operation of an imaginary counter-power. Clastres, for example, speaks at

length on the role of the chief. Contrary to popular images of these tribes, and baffling to early Europeans, chiefs were leaders without power: 'The chief is not a commander; the people of the tribe are under no obligation to obey. *The space of chieftainship is not the locus of power* […]' (Clastres, 1989: 206; emphasis in original). Clastres continues,

> The chief is there to serve society; it is society as such — the real locus of power — that exercises its authority over the chief. […] In a sense, the tribe keeps the chief under a close watch; he is a kind of prisoner in a space which the tribe does not let him leave.
>
> (Clastres, 1989: 207)

If these were societies *against* the State, then through what kind of mechanisms is the chief kept powerless? Anthropologist David Graeber, picking up on the thread left by Clastres, describes a peculiar attitude towards power prevalent in non-State societies:

> In egalitarian societies, which tend to place an enormous emphasis on creating and maintaining communal consensus, this often appears to spark a kind of equally elaborate reaction formation, a spectral nightworld inhabited by monsters, witches or other creatures of horror. And it's the most peaceful societies which are also haunted, in their imaginative constructions of the cosmos, by the constant spectres of perennial war. The invisible worlds surrounding them are literally battlegrounds. […] *It's not these contradictory impulses themselves which are the ultimate political reality, then; it's the regulatory process that mediates them.*
>
> (Graeber, 2004: 25; emphasis added)

It is this tortured nightworld of witches and, in an example Graeber describes later, the notion of 'flesh-debt,' that is asso-

ciated with the exercise of power. Those exercising power, in a sense, are believed to feed off of the bodies and spirits of those they dominate, to have obtained that power through evil deeds, through literally consuming the substance of others (Graeber, 2004: 27). And as Graeber suggests in the final sentence, it is the effect of this attitude towards power that ensures these societies fend off the emergence of power as an ongoing possibility. It is this imaginary world of witches and spirits that sets up the latent possibility of an altogether real revolt: the periodic witch hunts against those coming to exercise power over others, the killing of chiefs or their abandonment by their tribes (Graeber, 2004: 26-29; see also Clastres, 1989; Clastres, 1994). This is the imaginary counter-power: not the tortured nightworld by itself, but the role it plays in making these resistant acts against those in power a real, genuine and ongoing possibility. In putting down the possibility of power, Graeber claims this imaginary counter-power to also have a constitutive effect. He writes,

> Institutionally, counterpower takes the form of what we would call institutions of direct democracy, consensus and mediation; that is, ways of publicly negotiating and controlling that inevitable tumult and transforming it into those social states […] that society sees as most desirable […]
>
> (Graeber, 2004: 35)

Imaginary counter-power, therefore, comes to wield a double effect: both in opposing the eruption of power and in shaping the institutions of their own societies, backed in both instances by the ongoing threat of chieftal abandonment or death and their replacement with their own newly constituted social form.

§The abstract machine of the Exodus is a particular type of imaginary counter-power, one that operates according to a single category of resistance known as the exodus.[13] Drawing upon Michel De Certeau, we can categorise resistance practices into three distinct types according to the manner in which they act upon space and time. The first and archetypal form of resistance is the *strategy*. The strategy embraces all manner of practices that seek conquest over that which they resist. It is a resistance that is neither defensive nor sly, but stands firm and wages battle upon the same terrain as its object of contention. This is to say, it is the conquest of existing space and time. De Certeau wrote,

> I call "strategy" the calculus of force-relationships which becomes possible when a subject of will and power (a proprietor, an enterprise, a city, a scientific institution) can be isolated from an "environment." A strategy assumes a place that can be circumscribed as *proper* (propre) and thus serve as the basis for generating relations with an exterior distinct from it […].
>
> (De Certeau, 1988: xix)

The strategy is visible and proud. It seeks legitimacy through rightful conquest, and is founded upon the capture of territory and time, territory whose defence, enlargement and embrace will form its ongoing preoccupation. In Pierre Clastre's work, an imaginary counter-power built upon this model is one that threatens to rise up and reject the chief through death or otherwise.

Against the strategy is a second type of resistance, one without either proper place or time that we can call the *tactic*. The tactic is quiet, its existence being founded upon its invisibility. It

13. The Exodus, capital letter, refers to the abstract machine as a whole; exodus, lower case, refers to the event of desertion.

occurs in the interstices and cracks of power and bears a similarity to the art of jujitsu: with nothing of its own, its resistant practices must 'make do' with the objects and space of the hegemonic order; with nothing of its own it must turn these objects against themselves and must reinscribe them against their intended purpose. De Certeau wrote,

> I call a "tactic," on the other hand, a calculus which cannot count on a "proper" (a spatial or institutional localization), nor thus on a borderline distinguishing the other as a visible totality. […] It has at its disposal no base where it can capitalize on its advantages […]. Whatever it wins it does not keep. It must constantly manipulate events to turn them into "opportunities."
>
> (de Certeau, 1988: xix)

The tactic is a fleeting capture of otherwise occupied time and space, forced each time to start anew.

De Certeau suggests only these two categories of resistance. There is, however, a third type of resistance that depends upon an outside, a frontier, or the ability to turn an inside against itself. It goes by many names: the exodus, the line of flight, the refusal, the escape, and the fork. We can formulate this once again in terms of space, for while the strategy is the conquest of existing space, and the tactic is the momentary reclamation of official space and (mis)-appropriation of its objects, the exodus is the withdrawal from occupied space altogether. It is a kind of non-hegemonic resistance[14] that attacks that which it resists through an absence:

14. Non-hegemonic resistance is one that resists whilst not seeking to conquer and become the object of contention. See Deleuze and Guattari's notion of 'becoming-minor' (2004b: 320–322) and Richard Day's (2005) discussion of the concept.

it is a refusal to remain subject to a particular power and refusal to remain constitutive of that very power. En masse, the exodus deprives that which it resists of its constituent parts and of its subjects; it leaves it deserted. In the exodus, however, there exists not just resistance but possibility: the possibility that exists in the space to which they escape, the possibility of constructing something that avoids the perils of the past. In its desertion, the exodus always entails the creation of new relations and new forms of social organisation. The exodus, therefore, is a resistant practice that contains both destructive as well as constructive elements. In Pierre Clastre's work, the exodus is the abandonment of the chief by the tribe and the relocation of themselves elsewhere. Such a practice is made possible by the inability of the chief to deny such a course of action, by the presence of space elsewhere beyond its borders and, owing largely to an economic system based upon gathering and hunting, to the relatively low cost of such a move.

§Merging these two concepts into one gives us a first formulation of the abstract machine of the Exodus. The Exodus machine produces the prospect of the exodus as a kind of sword of Damocles: it forms an imaginary counter-power which, while imaginary, has effects that are nonetheless both real and constitutive of the social ordering. The Exodus requires a number of necessary conditions: first, the presence of space elsewhere, that is, an outside; the ability to leave for this outside, which is to say, a relative absence of constraint or coercive violence; a material abundance such that the space outside is not a space of certain poverty; and, finally, a discourse of counter-/anti-power wherein the exodus, now made possible, comes to wield its effects. Let

us turn, one by one, to each of these conditions of the abstract machine of the Exodus and their manifestation as the forking machine within the Compiz project.

The first element of the Exodus is the presence of an outside. Clastre's examples of chieftain abandonment, for example, were made possible by the space which existed beyond the domain of the chief. We can find a similar example in an article by Paolo Virno where he describes the practices of defection or 'the exit' that occurred during the industrialisation of the United States. He writes, 'one has only to think of the mass flight from the factory regime set in motion by the workers of North America halfway through the nineteenth century as they headed off to the "frontier" in order to colonize low-cost land' (Virno, 2003). The frontier and its low-cost land here act as the space outside that makes the very act of defection possible. In both cases, this 'outside' is quite literally a space beyond the grasp and embrace of a particular hegemonic machine, an outside that exists as the first and most important condition to establish the exodus as an ongoing possibility. The virtual space of Compiz can be thought in much the same way as the frontier: large hard disks, abundant network connections and a relative excess of computer power combine to form an outside that is both large and cheap. Defection is cheap but it is not, however, free: there remains a cost associated with this movement to an outside and with the creation of these new spaces, and this is just as true for virtual spaces as it was for the Amazonian defections and the mass flight of the factory workers. In satisfying this first condition of the Exodus machine, the costs associated with the movement to and creation of these outside spaces must be low enough that they remain a possibility.

The second condition of the Exodus machine is in the ability to leave occupied space, and this first and foremost means an absence of violence, latent or otherwise. It makes little sense to speak of the exodus-as-event as an ongoing possibility knowing full-well that any such escape would be met with violence and any movement to an outside space met with conquest or demolition. Nor does it make sense if the very act of escape is already foreclosed by the presence of borders and constraints that would deny such movement. Within Compiz, the possibility of violence was very much limited. Indeed, if we can speak of a virtual violence at all then it is in the 'flame wars' and 'denial of service' attacks, in verbal exchanges and technical sabotage.[15] These forms of violence, however, were absent from the Compiz project and there was otherwise no substantive threat of violence for defection. Constraint of movement, on the other hand, was the primary effect of the user-space machine whose mechanisms created fiefdoms of each of the various nodes of virtual space. This control, however, only extended so far as the node itself, and operated only so long as one desired to remain within its space. If one chose to leave its domain its mechanisms became powerless, having no effect outside its domain nor ability to restrict exit.

Having established the necessary conditions of escape, as we have in these first two conditions, does not mean that such a course of action becomes likely if it is one of guaranteed poverty. This is the third condition of the Exodus machine: the con-

15. A flame war is a textual exchange of heated, insulting and often vicious comments, usually on a public mailing list, meant to subdue opposition through shear intimidation. A denial of service attack is an attack upon a server by way of a network connection, wherein the server is overwhelmed by a deliberate flood of network requests.

dition of material abundance. Clastre's chieftain abandonment, for example, was made possible by the Amazonian tribes' means of production, namely gathering and hunting. In this, there was no major capital that was lost in desertion, and the ecology of the rainforest could provide roughly equally from place to place. Within Compiz, the prospects of escape would have been much reduced if escape entailed starting the coding effort anew and if, moreover, it entailed losing access to the very means of producing code such as programming environments and code management tools. As it turns out, it was the property regime of open source software in general, and Compiz in particular, that had the effect of satisfying this condition of material abundance. To understand this, we must make a digression.

To understand the relative novelty of the property regime of free and open source software, let us first concern ourselves with the machinations of proprietary software. To revise briefly some of what was discussed in Chapter One, Compiz-as-program was produced out of Compiz-as-code through a process known as compilation. That is, the code upon which the whole work of the project was directed was run through a compiler producing Compiz-as-program, a binary or mass of ones and zeros intelligible only to a computer. Unlike the source code, the binary cannot be edited and improved upon by programmers since they cannot understand its operations in the first place. Moreover, the process of compiling code into a binary is for all practical purposes one-way. This is the first technique of proprietary software: by distributing binaries of software whilst withholding the all-important source code, a company can foreclose the possibility of ongoing and independent development outside of its control. By this tech-

nique, the software is transformed into a pure end product and withdrawn from social production.[16]

But while the software is withdrawn from social production, the binary can nonetheless be easily copied and shared even if it cannot be edited. The sharing and duplication of the software outside the bounds of the originating company undermines its exclusive grasp on the product and it is compelled to orchestrate the necessary conditions so as to channel distribution through itself. There are technical means through which this can be done, such as using the techniques known as 'Digital Rights Management' or using product serials and keys to unlock products, with each of these implementing the abstract elements of the Passport machine to a greater or lesser degree. The most common method, however, is the use of the software license. The software license relies upon contract law, copyright statutes, patent protections and trade secret laws to specify the terms under which the software may be used, distributed and copied (Kim, 2008). In most cases these terms are highly restrictive and prohibit copying and distribution altogether as well as any attempts at reverse engineering or altering the software. Third party copying and distribution thus becomes illegal, the software becomes scarce, and legitimate sales and distribution are brought back within the purview of the company. In this way, a mixture of technical mechanisms and the latent violence of the State are enacted to create the binary as finite and scarce.

16. I owe my use of the phrase 'social production' to Yochai Benkler's *The Wealth of Networks* (2006). He elsewhere also uses the phrase 'commons-based peer production' interchangeably with 'social production.'

It is in light of these tactics that we can understand the practices of free and open source software. FOSS counters proprietary software on both counts: in the first instance it is founded around the open distribution not just of the precompiled binaries but the source code itself, and in the second instance FOSS projects use copyright law to guarantee recipients of code the right to modify and distribute it. The first of these countermeasures is straightforward: for software to be considered open source its source code must be publicly accessible. This is usually provided for download from a server either as a compressed file or by giving direct access to the code repository, as was the case with Compiz.

The second countermeasure is in the associated software license. Free software licenses make use only of copyright law, as opposed to using patents or contracts, and thus the license only comes into effect upon copying and distributing the software; it makes no prescriptions regarding its use. Compiz was licensed under three different software licenses. The bulk of its code was covered under the Massachusetts Institute of Technology or MIT license, the remainder, notably the 'Gnome decorator,' was released under the GNU General Public License (GPL) and a small number of files under the GNU Lesser General Public License (LGPL), which we shall not cover here. The two main licenses represent two very different philosophies of free software. The MIT license grants unrestricted rights to copy, modify, merge, distribute and sell copies of software to which it is applied on the condition that the copyright notice is distributed alongside those derivative works (Open Source Initiative, n.d.). There is no requirement for derivative works of software to be similarly open source, and the terms of the license only apply to the section of copied source code and not to any resulting binaries or additional

code. The MIT license, therefore, allows software under its domain to be integrated into closed source projects. This is known as the 'permissive' aspect of the license and it is the primary reason why MIT-licensed software is commonly embraced by commercial enterprises. Aside from its requirement to distribute the license alongside derivative works, the MIT license has an effect very similar to deploying no copyright at all.

Like the MIT license, the GPL grants similar rights to copy, modify, merge, distribute and sell software released under its license. Where the GPL differs, however, is in its treatment of derivative works. The GPL specifies that the right to modify software under its license must exist for all derivative works. This requirement therefore necessitates that all future versions of the software must remain open source. Secondly, unlike the MIT license, additions and modifications to GPL-licensed code that cannot be 'reasonably considered independent and separate works in themselves' also come under the license (GNU Project, 1991). The effect of this aspect of the license means that the GPL comes to embrace not just the original code itself but the entire body of code that forms a single piece of software, an effect that some critics have described as 'viral' in nature (Mundie, 2001). Where the MIT license places few constraints upon its derivative forms, the GPL remains in full effect upon the growing body of derivative code for the life of the copyright, and this is known as its 'restrictive' aspect. The overall effect of the GPL, then, is to ensure code that comes under its purview from ever becoming closed source and from ever being released under terms that would otherwise restrict its distribution. GPL-licensed software becomes open to ongoing social production for the life of the copyright,

and this inversion of the normal role of copyright is popularly known as 'copyleft.'

Quite in contrast to the measures deployed by proprietary software makers to transform their code into a pure end product, these two counter-measures — the open source code and the inverted copyright — combine to ensure that code produced by free and open source projects remains open to ongoing social production. They work to ensure that there exists the potential for code to be edited and modified by others, transformed and distributed elsewhere, and that the code itself can become the basis for other programs. We can call this the socialisation of the code.

In addition to the code itself, the means of production such as the GNU C Compiler, the code editors such as VIM or EMACS, the CVS code repository software and the various GNU/Linux components to test and run Compiz-as-program, similarly existed in fully socialised forms. These tools, therefore, could be duplicated and transferred just as the code produced within Compiz could be easily and trivially replicated.

To return from our digression, then, we can say that the socialisation of both the code and the means of production within Compiz and throughout the free software community in general produced a situation of material abundance, one that guaranteed that a fork from Compiz was not doomed at the outset to code poverty. The potentiality of the exodus-as-event was therefore not made undesirable by material constraint and in this manner the third condition of the Exodus machine was satisfied.

The fourth and final condition of the Exodus machine is a discourse of counter-power. The previous conditions produced the exodus-as-event as both a potential and, in the last instance,

as a potential made not undesirable by poverty. But these conditions, though perhaps satisfied, may languish unbeknownst if not animated and given life through the ongoing work of a discourse of counter-power, and it is in this discourse that the Exodus machine wields its constitutive effect. By discourse I mean literally the words and stories that are told that allude to the exodus, as well as the practices that in some fashion embody that possibility. In the example by David Graeber that we saw previously, this discourse of counter-power lay in the tortured nightworld of the witches and their flesh eating deeds, and in the necessity for their opposition. Constantly reiterated and retold, these stories made clear what lay before those who crafted for themselves positions of power, the potential that lay in wait, and in this manner this discourse came to constitute their social organisation as predominantly non-hierarchical.

Within Compiz, too, there existed a discourse of counter-power, one that constantly reiterated and threatened the potentiality of the fork. The project was haunted by a 'spectre of division,' but in a rather different way than Clastre's spectre of division between the dominating and dominated. From the very outset, ruminations abounded of the possibility of a fork, some of which we have already encountered. Quinn Storm's early email in response to the rejection of the window shadowing patch raised the possibility of the fork, though she noted at the end 'I wish this does not have to happen, and hope it does not' (Storm, Quinn: 2006-05-03 09:39). We have previously understood this exchange in terms of simply raising the possibility of the fork but we can, now, understand this as also contributing to an ongoing discourse of counter-power, as reiterating the spectre of division. Moreover, her final sentence is both a threat and an implicit plea

that David Reveman (we presume) better attend to the wishes of the Compiz community.

One of the more lengthy exchanges around the possibility of the fork occurred during the second half of June, still some three months before the fork occurred. It was Guillaume's email that triggered this discussion which, if we recall, asked David for greater engagement with the Compiz community, to provide explicit standards and coding styles, and to communicate better his goals with regards to the project (Seguin, Guillaume: 2006-06-24 16:24). The first reply to this was from Wulf C. Krueger who, noting that these issues had been ongoing, replied simply, 'If I were you, I'd just branch "officially" and compete' (Krueger, Wulf C.: 2006-06-24 17:20). Quinn Storm, who was next to reply, wrote a substantial response. In this she wrote,

> I've wanted to avoid an "official" fork as long as possible, feeling that it could in the end work against everyone's best interest, but this all depends on the upstream (freedesktop.org/novell) developers. […] If it comes down to it, in the spirit of the GPL [GNU General Public License], I am not against managing my tree as a semi-fork (I'd still sync with updates from freedesktop cvs of course, as davidr and friends often commit important updates).
>
> (Storm, Quinn: 2006-06-24 23:04)

Once again, the prospect of the fork is in a sense used as a bargaining piece. She suggests her aversion to the fork but signals her willingness to proceed if the upstream developers of Compiz do not change their behaviour. It is interesting, too, to note her portrayal of the GNU General Public License as not simply allowing for the possibility of modifying and distributing code apart from

the main project, but that the possibility of forking in some sense embodies the 'spirit' of the GPL. In a subsequent email Matthias Hopf, a close associate of David, advised against a fork, suggesting that 'Branching has always been the source for problems' (Hopf, Matthias: 2006-06-26 03:35). This prompted a discussion around the merits of forking, with Wulf C. Krueger responding to Matthias by citing some particularly famous forks. He wrote, 'Tell that [to] Emacs/XEmacs, egcs/gcc or XFree86/X.orgX11. :-)' (Krueger, Wulf C.: 2006-06-26 11:21). Matthias replied,

> Yes, and all of them have been a [pain in the ass], especially emacs/xemas, because both are used. The egcs split turned out irrelevant. The X.org split turned out good, but only because almost all developers switched side.
>
> (Hopf, Matthias: 2006-06-28 02:14)

We can see that the discourse of forking exists not just within the confines of the Compiz community, but as shared stories of other free and open source projects. Moreover, I would suggest that the history of many of these forks could be considered common knowledge, as well as the knowledge of the purported causes that led to their forking. These stories were written upon objects outside of the Compiz project proper, on blogs, websites, mailing lists and in a number of books documenting the history of free software.

In addition to the ongoing ruminations around forking and the common knowledge and stories of historical forks of other projects, there was at least one more aspect to the discourse of counter-power within Compiz. This existed not as a literal discourse, but was embodied in the existence of the Quinnstorm branch. At the founding of the mailing list and the very first

emails, the Quinnstorm branch was already in existence having been created shortly after the initial code of Compiz was made public in early 2006. Throughout the course of the following months, the Quinnstorm branch was kept synchronised with the official Compiz code as well as progressively including plugins and other code of its own, code that for one reason or another was rejected or simply omitted from Compiz itself. During this time, the Quinnstorm branch was ostensibly and only a branch: a parallel effort that viewed itself as part of and contributing to the Compiz project. But it was also a threat. In many ways the fork had already occurred: the production of new space on a new server, the duplication of the Compiz code, the setup of the tools required for code production, and the fostering of a community around the Quinnstorm branch. With this substantial work already completed, what remained for the fork was more political than anything else: the declaration of the fork, the naming of the new project and the institution of their own practices of collaboration as separate from Compiz. The Quinnstorm branch made the potential of the fork considerably more real, and the ongoing work around it surely contributed to the discourse of counter-power within Compiz.

These four conditions for the abstract machine of the Exodus — space elsewhere, the ability to leave, material abundance, and a discourse of counter-power — were each instantiated within Compiz as the forking machine. Though it would eventually culminate in the event of the fork itself, it would for the greater part of 2006 instead wield a constitutive effect upon the project.

§We saw in the last chapter that the Passport machine produced David Reveman as the super-user and gatekeeper of the Compiz project. We also touched upon the precarious nature of the power derived from these roles. The effect of the imaginary counter-power of the fork was the production of this precariousness, and it was to produce him, in addition to gatekeeper, as maintainer. 'Maintainer' was his designation within the project, and it is a name that is commonly used across free and open source projects. Unlike the role of the gatekeeper which expanded one's exercise of power, the role of maintainer burdened one with a range of responsibilities. Much like Clastres' chiefs, the maintainer becomes a kind of prisoner to the project. This was the primary constitutive effect of the forking machine.

The role of the 'maintainer' is the most common designation amongst free software projects for the lead developer or developers. Its distinction from the role of the gatekeeper lies in its relation to the code. While the gatekeeper role was founded upon the exclusive control of a single instance of the code, the maintainer role was founded upon the ongoing stewardship of the code, where one became a kind of caretaker or custodian. This included jobs such as patching bugs, adding features, developing code, accepting the patches submitted from others, providing for the necessary infrastructure of code repositories and so forth. All the day-to-day work required in developing the code that we saw in Chapter One was, in the last instance, the duty of the maintainer.

The role of the maintainer was a job that was imposed from outside. It was intimately tied to the role of the gatekeeper and formed a kind of bargain or exchange. This exchange went such that the community of developers chose to recognise the code

under the control of the gatekeeper as the official and proper code of the project, and in return the gatekeeper became required to adequately maintain this code and the project. Such an exchange depended upon the ongoing possibility of the fork, for in the fork contained the possibility that a part of the community of developers may instead choose to recognise a copy of the code held elsewhere. That is, the event of the fork, in addition to being a mass defection, is also the establishment of a competing claim as to the *proper bearers* of the code. In choosing to acknowledge David Reveman as the proper bearer of the Compiz code, therefore, he was expected in return to perform the role of maintainer.

Against the potential despotism of the gatekeeper, the imposition of the role of maintainer produced an opposing, anti-authoritarian force. Its locus lay not within the gatekeeper but dispersed within the community of contributors and its effect, by raising the ongoing possibility of rejection-through-forking, was to bind the maintainer to their duties. The gatekeeper was imposed with the task of maintainership much like Clastres' chiefs became prisoners to their societies. Of this relationship Clastres writes,

> The second characteristic of the Indian chieftainship — generosity — appears to be more than a duty: it is bondage. Ethnologists have observed among the most varied peoples of South America this obligation to give, to which the chief is bound […]. And if the unfortunate leader tries to check this flight of gifts, he is immediately shorn of all prestige and power. […] Greed and power are incompatible: to be a chief it is necessary to be generous.
>
> (Clastres, 1989: 30–31)

For all the potential powers that may be exercised as part of the gatekeeper role, David Reveman was carefully bound in these capacities. The vicarious powers of the gatekeeper role could only be exercised so long as the community continued to desire to participate, a desire that was tied to the ongoing status of the project as the proper bearer of the Compiz code and him as its maintainer. The maintainer was conferred a status of prestige within the group, he was enabled to exercise a limited power over both the ongoing development of the code and the community of contributors, but in exchange he was bound to his duties as maintainer, duties that should he have failed to perform he would risk finding himself quickly and promptly abandoned.

The nature and content of these duties was formulated by the community of contributors and was the subject of much debate. In general, it was formulated as part of the discourse of counter-power, formulating the expectations of maintainership alongside the possibility of the fork. In the now-familiar discussion around coloured window shadows, for example, a contributor named David Rosenstand replied to Quinn Storm and her objection to David's authority in determining which features were 'useless crap.' He wrote,

> Adding code for options that nobody wants to use doesn't make sense. The maintainers will have to maintain more code, and the users will have a harder time finding the useful ones and potentially discover (and report) more bugs.
>
> […] This critique seems unfair. "David's vision" is just responsible maintainership.
>
> (Rosenstand, Mark: 2006-05-08 02:37)

Elsewhere, another email talked of the possibility of 'spaghetti code' (Liebetraut, Thomas: 2006-06-25 06:11), and that David's decision here to reject the window shadowing patch was, in fact, appropriate in service of his other responsibilities as maintainer. There is in these disagreements something of an attempt to formulate and clarify what are reasonable expectations to have for a maintainer and, in the same instance, to judge David against these expectations. In another instance, a contributor claimed that David had failed to properly communicate with the group and this email was met with agreement from others on the list. Matthias Hopf, however, replied

> David is typically producing code. Lots of high quality code. If he were chating [sic] as much as others (including me) do, compiz wouldn't be where it is now.
>
> (Hopf, Matthias: 2006-06-26 03:35)

Once again, there is an articulation of the responsibilities of the maintainer, here both to be communicative as well as to continue development of the Compiz codebase, but there is also an acknowledged trade-off in these different obligations, an acknowledgement that the maintainer only has a limited amount of time. Thus even as the maintainer becomes a kind of prisoner to the group there is also a discourse that seeks to articulate the 'reasonableness' and fairness of these obligations.

§ For all the effects that the forking machine had upon the dynamics of the group, it nonetheless came about that the Quinnstorm branch was declared a fork proper and renamed as Beryl on 18 September 2006. The fork, however, was not a total abandonment of David Reveman. Many believed that he was

indeed properly performing his duties as maintainer and, in the last instance, producing better quality code than that which was being added to the Quinnstorm codebase. The contributors who moved to create the Beryl project obviously disagreed and in the official Beryl announcement attempted to list the different reasons for the fork. As the first of these reasons, the Beryl project claimed David to be unresponsive to patches being submitted:

> Lots of people suggested to send our patches to the mailing list. […] Furthermore, it's really unsure that David would happily accept these patchs [sic]. I'm even nearly sure that most of them would be rejected. Check the Xinerama issue; David is [only] willing to implement his own stuff.
>
> (Seguin, 2006)

Second to the list of charges was the problem of communication, where they argued that David was unresponsive and cited the relative inactivity of the Compiz mailing list compared to other Compiz forums, and that he had 'never really published what he was intending to do and implement on a long term plan' (Seguin, 2006). Forking, they argued, 'gives us the opportunity to introduce our own roadmap, our own goals, our own release cycle' (Seguin, 2006). In addition to these issues with David's role as maintainer, they claimed a number of technical reasons for the fork. The first, and the most important that they perceived, was the divergence in code that had slowly occurred as the Quinnstorm branch had accepted patches whilst Compiz had not. The announcement read,

> During this summer, and during the last few weeks, some major additions were done in compiz-quinn-

> storm […] Consequently, we reached a situation where it's quite impossible to come back.
>
> (Seguin, 2006)

The announcement also indicated that there was confusion 'downstream' about which code was the official Compiz code, and that for the sake of the various downstream Linux distributions it was best to fork. The final part of the announcement insisted that the fork was amicable:

> Finally, please note that this is a friendly fork. We don't have anything against David, and we understand that his hands may be tied due to his work at Novell. We just need more freedom. Thanks David for the wonderful job you did. We'll just try to keep the quality level you introduced.
>
> (Seguin, 2006)

The rest of the announcement detailed the practical work that was to be done to carry out the remainder of the work required of the fork.

The claim of the 'friendly fork' was dubious. Frustrations amongst the forkees were obviously high enough to justify the fork itself, and in an email a few months after the fork David expressed his own frustrations concerning the fork, claiming it was unjustified (Reveman, David: 2007-02-16 08:06). Even so, a flow of code continued between the two projects. Patches to Compiz core were also often applied to Beryl. In the opposite direction, Mike Dransfield created a third-party repository of plugins that were originally sourced from Beryl but had been tweaked so as to work in Compiz too. This was eventually 'packaged' as 'Compiz-extras' (Hopf, Matthias: 2006-10-20 05:35). Moreover, in recognition that there was much to be gained from sharing code in this

way even after the event of the fork, many contributors talked at length of ensuring compatibility between the two projects. Mike Dransfield wrote, for example, that failure to ensure this compatibility would mean that 'plugin writers are going to have a harder time in the future to make their plugins compatible with each fork,' and that 'there is clearly demand from the "community" for [compatibility]' (Dransfield, Mike: 2006-10-06 12:04).

The fork itself appeared to have a number of effects, though whether these were caused by the event of the fork or were merely coincidental is difficult to discern. A week after the fork, David created a plugin template which allowed for the easy creation of plugins that also adhered to the coding styles that he was enforcing upon the codebase (Reveman, David: 2006-09-27 11:19). A further week after this, David released the much-requested coding style guidelines (Reveman, David: 2006-10-05 13:12). And, on 15 November, David proposed a detailed project roadmap, one the key reasons given for the fork (Reveman, David: 2006-11-15 08:26). In general, there appeared a marked change in David's behaviour *after* the fork. He began to comply with many of the stated reasons for the fork and became much more communicative on the mailing list. One contributor, Shawn Starr, commented on this change:

> I am glad that your [sic] spending more time on compiz now and are being responsive to people. […] I guess in some ways, the fork has induced change in compiz and that was really the idea.
>
> (Starr, Shawn: 2006-10-06 12:54)

Subsequent to the event of the fork, therefore, a machine bearing great similarity to the forking machine came to wield an effect upon the project. But it was no longer the *latent* possibility of the

fork that was the force behind this effect. Rather, the Beryl project represented a fully constituted counter power and by its very existence it raised the ongoing prospect of defection, threatening to lure away the remaining developers. In this manner it appears to have further bound David Reveman to his duties as maintainer within Compiz.

§Though the project was to eventually fork, the forking machine had a considerable effect upon the organisation of the Compiz project, both before and after the event of the fork. Its first three elements — the presence of an outside, the absence of violence and restraint, and the material abundance offered by the socialisation of code that is unique to free and open source software — produced the prospect of the fork as a genuine possibility. Its final element, the discourse of counter-power, was to make this prospect known and to animate the forking machine. In this discourse, the possibility of the fork was reiterated such that the gatekeeper came to be imposed with the additional role of maintainer. This role was the outcome of an exchange, one in which the community of contributors granted to David Reveman the status of proper bearer of the code replete with its gatekeeping powers but, like Pierre Clastre's chief, in this exchange David became a kind of prisoner to the group and became bound to his duties as maintainer. These duties were the constant subject of discussion, concerning both the expectations others had of him in his role as maintainer and the fairness of these demands upon his work.

V. The Module

In which our adversary Complexity finds himself tamed by a happenstance of objects, borders and documents

In 1975, Frederick Brooks wrote the iconic book *The Mythical Man Month* on his observations of the organisation of software production. In this he wrote of one of the key difficulties facing not just software production, but any sufficiently complex project:

> The dilemma is a cruel one. For efficiency and conceptual integrity, one prefers a few good minds doing design and construction. Yet for large systems one wants a way to bring considerable manpower to bear, so that the product can make a timely appearance. How can these two needs be reconciled?
>
> (Brooks, 1995: 31)

The problem was the practical task of *working together* on a project that was large in the number of its participants, large in size, and both delicate and difficult in operation. The solution, Brooks wrote, was to pursue what later became known as the 'Cathedral' model. This was a model where the architecture of a project would emanate from the mind of just one person — thus guaran-

teeing 'conceptual integrity' — and which would overcome the problem of complexity and an excess of design ideas by enforcing a unidirectional flow of communication, from top to bottom. The problem of complexity was to be solved with the stamp of absolute hierarchy. Brooks would write that '[this] is an autocracy that needs no apologies' (Brooks, 1995: 46), but it was also an autocracy that simply could not exist in free and open source software, lest a project immediately face the prospect of a fork.

In 1997, Eric Raymond wrote *The Cathedral and the Bazaar* which famously documented an alternative solution to Brook's problem that was then in operation in the free and open source community. Raymond, himself a maintainer of a FOSS project, wrote,

> I […] believed there was a certain critical complexity above which a more centralized, a priori approach was required. I believed that the most important software […] needed to be built like cathedrals, carefully crafted by individual wizards or small bands of mages working in splendid isolation […].
>
> Linus Torvalds's [the originator of the Linux kernel] style of development — release early and often, delegate everything you can, be open to the point of promiscuity — came as a surprise. No quiet, reverent cathedral-building here — rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches […] out of which a coherent and stable system could seemingly emerge only by a succession of miracles.
>
> (Raymond, 2000)

This 'succession of miracles' owed its greatest debt to the architecture of the code and the Linux operating system as a whole, an

architecture that from top to bottom implemented the machine of the Module. This was an architecture of decentralised and interdependent code that 'talked' to one another using standardised protocols, inherited in large part from the architecture of Linux's predecessor, Unix, which was created in 1969. The architecture was at once technical and social, its architectural shape lending itself to a correlating shaping of social relations. Three effects of the machine of the Module stand out. Firstly, the modularisation of the code into discrete segments produced a correlating division of tasks which in turn lent itself to a particular type of division of labour: one that allowed for a great deal of autonomy and, as a result, allowed for a massively parallelized undertaking — potentially without anything of Brook's autocracy. Secondly, the task of creating and maintaining order was delegated to the architecture of the modular system itself. Put differently, the task of stabilisation was transferred from the person of the 'system architect' to both the space of module and object of the protocol standard. Questions around the ordering of social and technical relations therefore came to be addressed to these objects. Finally, the layers of modularity, from the level of the code on up to the largest structures of the Linux operating system, produced a far-reaching social order whose structure is best described as a type of anarchistic federation.

In this chapter we will delineate the elements of the abstract machine of the Module by tracing the development of one of its archetypal forms, the System/360, before exploring its concrete instantiation within Compiz and its effects upon the ordering of the project.

§ Despite advocating the cathedral model for software design, Frederick Brooks ironically oversaw the development of one of the earliest and most celebrated modularised artefacts, a computer produced by IBM called the System/360 in 1967. The process leading to the development of the System/360 spanned more than twenty years and is testament to the difficulty of appropriately modularising a complex artefact. In this process we shall come to see the elements of the abstract machine of the Module.

The first computers built during the early 1940s were thoroughly interconnected, and the tasks of designing, producing and using a computer overlapped with one another. It was only upon seeing these early computers in operation that it became possible to start conceptualising them as combinations of discrete functions. The first of these attempts was a memo issued in 1946 by Arthur Burks, Herman Goldstine and John von Neumann (BGV) which specified the different functional components of the computer with which we are still familiar today — memory, processor, input and output devices, and secondary memory or storage — as well as a separation between the computer design (hardware) and its use (software) (Baldwin & Clark, 2000: 155-157). The memo fell short of describing true modularisation, but it was nonetheless an important milestone: rather than conceiving of the computer artefact as a single integrated mesh of parts, it began to conceive of them as distinct functional components and, as a result, it could talk about the engineering problems unique to the design of each of these individual components (Baldwin & Clark, 2000: 156).

The BGV memo was an attempt at a mental decomposition of the emerging artefact of the computer. The physical reality of the computer, however, remained a thoroughly integrated and soldered mess of parts, and each new computer had to be designed anew. The BGV memo had made tentative steps towards the standardisation of some of the design rules of making a computer, such as the binary encoding of instructions, and in 1948 IBM produced the first standardised circuit, a 'pluggable unit' that could be inserted and removed from the rest of the computer. Though the rest of the computer remained largely integrated, this unit was truly modular: it was self-contained, of a standardised size and provided a standardised interface in the form of its connecting 'plugs' (Baldwin & Clark, 2000: 162).

With the introduction of the transistor-based circuit replacing the vacuum tubes of old, IBM attempted early on to standardise its form more rigorously than it had the pluggable circuit, which despite all attempts had grown in complexity and proliferated into over two thousand different combinations by 1957. Thus, the Standard Modular System (SMS) was introduced in 1958 prescribing a set of restrictive rules for transistor-based circuit design and manufacturing (Balwdwin & Clark, 2000: 163). Like the pluggable unit, the SMS decomposed the circuit into numerous smaller functional elements, each carefully and thoroughly prescribed in terms of size, its materials, its interconnections and so on. Crucially, the SMS introduced a policy of 'information hiding.' Information hiding concerned the interiority of each element, whereby the knowledge of how an element was internally organised was not only unnecessary to other elements wishing to communicate with it, but that this ignorance was enforced as a matter of policy. From the outside, each element appeared as a

black box. This design policy of information hiding was coupled with the creation of an interface for each unit, wherein a simplified façade was constructed which put forth a limited number of standardised 'hooks' for allowing communication between elements. Each component of the SMS therefore provided an interface that allowed for a set of limited and simplified interactions whilst separating and hiding its actual internal implementation, a process also known as abstraction.

The impetus behind the slow decomposition and modularisation of the computer was the desire to achieve high volume and low cost manufacturing through the standardisation of parts and their concurrent production. This is to say, modularisation was primarily driven by the interests of Capital. IBM therefore desired to apply the same techniques of modularisation that had been applied to the transistor circuit in the form of the SMS to the totality of the computer. Having now had some twenty years experience with the computer, and having now come to understand the different elements that were involved and which functions were the same across all computers, it was now possible to formulate a set of encompassing 'design rules.' The goal of IBM's System/360 project, as outlined in what became known as the SPREAD report, was to create a family of computers that for the first time fully embodied these rules, and which were to be fully standardised, modularised and compatible with one another. Leveraging the modular design, the report proposed three design phases. There was to be a design rules phase, which would intricately detail the allowed interactions between the modules, a parallel work phase in which each module would be independently developed, and finally an integration and testing phase in which the modules would be combined. By 1967 and under the

management of Frederick Brooks, the System/360 was completed spanning 50 new pieces of hardware each developed and manufactured in parallel according to the design rules that had been laid at the outset of the project (Baldwin & Clark, 2000: 169–192).

§ The System/360 was the first, most complete instantiation of the abstract machine of the Module. The apparent simplicity of the modular computer came from a twenty year decomposition of the complex, integrated computer, during which time the architects of the computer came to understand their own creation. Drawing upon this history, we can identify three primary elements of the machine of the Module: the module, the interface, and the standard.[17]

The module is a particular spatialisation of a complex artefact. It is the creation of a number of bounded spaces or 'interiorities,' each space being accorded a particular function or domain of tasks. The ideal spatialisation is one where the entirety of a particular function occurs within its functional space, a feat that requires a full appreciation of the different functions an artefact will be expected to perform and their relations to one another. The interior of each of these bounded spaces is hidden from the spaces beyond its boundary either through physical or technical impediment, or an adherence to a policy of information hiding. Each module therefore comes to resemble a black box, a discrete

17. Though not directly cited here, I must acknowledge Narduzzo and Rossi's 'Modularity in Action' (2003) in helping me formulate these abstract relations of the Module.

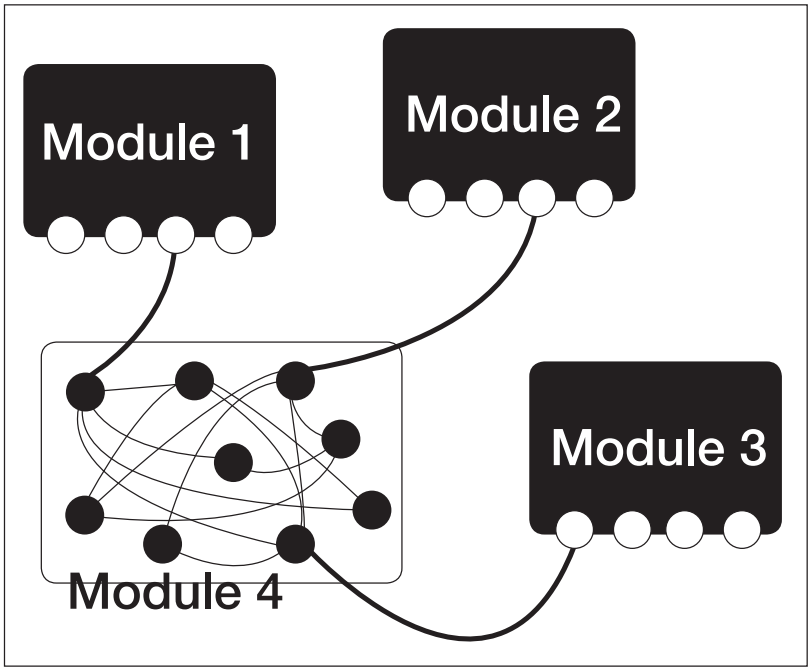

Figure 5. A simplified depiction of a modularised artefact. Each modular space is dedicated to a single function and is carefully bounded from the others. From the perspective of Module 4, the other modules appear opaque or as 'black boxes,' presenting just a few simplified hooks as part of their interfaces. Note that relations within a module are complex and many, but between modules they are few and simple.

entity whose inner complexity is hidden and whose multiplicity is subsumed to a singularity.

The means of intercommunication between modules is via their interfaces, these being the surfaces of their bounded spaces. It is upon the surface of the interface that the module exposes a limited and simple set of 'hooks' which accept communication from other modules, and which proceed to translate and pass these communications on into the interiority of the module. These hooks, for example, correlated to the plugs of IBM's pluggable circuit or to the 'pins' of the SMS. The interface, as a

mediator for communications and requests coming into a module, abstracts away the complexity of the module's own internal machinations. This process of abstraction allows for the module to maintain a relatively stable surface that changes little or not at all, to present an exterior that, from the outside, remains constant, whilst retaining the ability to change the means through which these intermodular requests are implemented.

A modularised artefact above a certain complexity typically requires several tiers of modular spaces. In such cases, modules are arranged hierarchically, resembling a Matryoshka doll, where each module is potentially itself decomposed into further spaces of interiority. Those modules at the topmost tier have the broadest functional definitions and present the simplest and most abstracted interfaces. In turn, they are internally organised into modular spaces of more specific and less abstracted functions. In such modular hierarchies, the rules of information hiding remain in force, and modules nested within larger modules inherit all the limitations of their ancestor elements.

The final element to the machine of the Module is the standard. The standard exists primarily in written form in documents and in reference manuals, and it is known variously as the specification, the documentation, the protocol, or simply as the standard. It concerns itself primarily with the question of boundaries. In the first instance, it concerns itself with the *location* of these boundaries, namely, the division of an otherwise continuous space into the discrete spaces of the modules. The standard describes the manner in which an artefact undergoes modular decomposition, and accords to each a specific functional task. In the second instance, the standard concerns itself with the *interac-*

tion between modular boundaries, that is, their interfaces, both in terms of the functionality they must or must not expose and the methods through which this functionality may be requested by other modules. As we saw with the history of the System/360, the standard or design rules are the most difficult and most contested aspect of modularity for it is this aspect that has the greatest ramifications. They are difficult, for the standards must remain relatively fixed for some time. Both the spatial decomposition of the artefact and the specification of interfaces, therefore, must be of such a design that they can both allow for a well functioning system in the present as well as accommodate future changes. This is primarily a technical question. The standards are also a site of contestation for they establish the constraints and possibilities of working with such an artefact. That is, the standards concern both the product itself *as well as its production*, both artefact and process, and this is primarily a social and political question.

§Let us first concern ourselves with the product. The abstract machine of the Module was instantiated at virtually every level within the Compiz project as well as throughout the ecology of which Compiz was a part. At the level of Compiz-as-code there were three tiers at which modularity was operating: at the level of the function, the file, and the plugin. Above this, there existed a fourth tier in which the entirety of Compiz formed a single modular space, interacting alongside other programs and libraries.

The two lowest tiers of modularity were only partial in their instantiation. At the lowest level a number of functions encapsulated discrete sections of code, hiding the code contained

within from outside and allowing for it to be triggered only by way of calling upon the function's designated name. The file operated one level up, and grouped together similar functions of which only a few were made visible from beyond its boundary, thus functioning as a de facto interface. These two lowest tiers of modularity worked to simplify and organise the code. Each implemented two of the three elements of the Module: both implemented the hidden interiority of the module and both erected a simplified interface. Neither, however, produced anything of a standard. At these lowest levels the code changed to such a great degree from day to day that the rigidity of a standard was largely impossible and would only have been an impediment to ongoing work. In place of the standard, therefore, the development of these lowest levels of code formed the ongoing preoccupation of discussion on the Compiz mailing list, down to details such as file names, function implementation and, as we have seen in previous chapters, code style.

The plugin architecture sat one level above the file. This architecture was built upon the construction of 'Compiz core' which, as the name suggests, implemented the most fundamental aspects of the Compiz program. Compiz core implemented functionality that was essential to the proper running of Compiz as well as acting as a central reservoir for functionality so common that it would otherwise be separately and repeatedly implemented elsewhere. By itself, Compiz core did very little. The majority of the visible features of Compiz were instead implemented by bounded modules of code — 'plugins' — that sat outside the boundary of core and which could be easily inserted and removed from Compiz. The plugins had as their building blocks the hooks provided by the Compiz core interface, known as the Application

Programming Interface (API). These hooks included basic drawing instructions, for example, or functions which detected user input. Unlike the functions and the files, the plugin API was required to be stable, predictable, and was haphazardly and loosely documented on the Compiz wiki, on various websites as 'how to' guides and in the source code itself (for example, Woodhouse, n.d.; Anon., 2008a). Additionally, several existing plugins were held as examples of good plugin design (Anon., 2008b) and others — 'dummy plugins' — were written simply for educational purposes (Dransfield, Mike: 2007-01-04 07:02).

The architecture of Compiz was not static: there was an ongoing process of re-modularisation, properly known as 'refactoring.' Remodularisation was part of the ongoing process of coming to understand the functionality of an artefact, or coming to discern where modular boundaries should lie, and how interfaces should function. It was also directly related to the ongoing development of the code. Remodularisation, for example, involved splitting a single module into two or more functionally distinct units, or moving a small amount of functionality to a different modular space to which it was better suited. On 4 October 2006, for example, David Reveman was reviewing a patch to a plugin and noted,

> Looking through the code quickly I found that some code from the minimize plugin has been duplicated. We might want to consider sharing some of that code by putting it in the core.
>
> (Reveman, David: 2006-10-04 09:10)

Such disruptions to the modularisation of the artefact were often handled through a technique known as versioning, whereby the ongoing evolution of the object Compiz-as-code was arbitrarily

demarcated at certain junctures, assigned a version number, and its modular architecture stabilised.

At the highest tier, the Compiz project and its code was nested within an ecology of other software projects. As we have moved up the modular hierarchy, from functions, to files, and to plugins, there has been a correlating increase in the use of standards. At this highest level, the use of standards and specifications became both more verbose and more stringent. Compiz-as-program operated closely with a number of other programs and its interactions were mediated by each of their respective standards documents. Most importantly, these included the X Window system and both its Extended Window Manager Hints (EWMH) and the Accelerated Indirect GLX (AIGLX) specifications, the various video drivers and their standardisation in the Open Graphics Library (OpenGL), and the underlying operating system and its Portable Operating System Interface for Unix (POSIX) standard. For each of these components and their standards, Compiz was built both expecting their compliance and, simultaneously, was itself expected to comply.

Compiz was thus composed of a series of nested modular layers, from the lowest tiers of the function and the file, to the spaces of plugins, and to the highest tier where Compiz was itself a modular element alongside other programs. At each layer, modules adhered to policies of information hiding, they implemented interfaces of various kinds, and, moving up the tier, progressively implemented more verbose and stringent standards. This modularisation was at once a technical organisation of the product and, simultaneously, a social organisation, concerning both product and production. It is to this latter aspect that we now turn.

§There were three principal social effects of the modularisation of Compiz, the first of which concerns the autonomous nature of labour. The modular decomposition of the Compiz artefact lent itself to the production of a 'functional' division of tasks and this, combined with the explicit statement of intermodular dependencies contained within the standards, enabled a type of labour that was of a highly autonomous character. This stands in marked contrast to another type of division of tasks, one that we can call 'mechanistic,' which produces a type of labour that is instead highly dependent, constrained and static.

Let us first recall the manner in which a modular artefact is decomposed. The aim of modular decomposition is to divide the artefact into units that are of relative independence to one another, where the majority of the workings of each module remain within its bounded space, and where communication between modules occurs in but a few, highly standardised forms. There is an inverse relationship between the number of dependencies and their frequency of use: a well-modularised artefact is one where relations *within* a module may be both numerous and dense, even though any single linkage may be employed rarely, and conversely where the relations between modules are few, though well trodden. To achieve such an ideal organisation, the choice of modular borders and the arrangement of parts is guided, above all, by their perceived role within the artefact as a whole. That is, by their *function*. The artefact as a whole must be designed and decomposed or, rather, designed *to be* decomposed, so that each of its modules implements, to the largest degree possible, the totality of their designated function whilst simultaneously limiting the dependencies between modules only to those that are the proper function of a module elsewhere. This is no easy feat. As

we saw with the development of the System/360 and, indeed, the ongoing re-modularisation of Compiz, the decision of how to characterise and distribute the functions of an artefact is both difficult and uncertain, and oftentimes is made clear only after having observed the workings of the artefact

This modular decomposition of Compiz-as-code produced in the same moment a corresponding division of tasks. When one approached the Compiz artefact, one was not confronted with a dense and intermeshed object, but with an object that had already undergone modular decomposition. One was presented, that is, with a series of spaces already constructed and oriented towards specific functions. The labour process was similarly spatially divided. The pursuit of implementing a piece of code generally meant one was directed toward the corresponding functional space, a space from which one generally did not have to stray. The different tasks required to implement window transparency within Compiz, for example, were entirely bound within the domain of a single plugin. Tasks were thus grouped together based upon their ends, based upon the functionality they sought to implement. This was, therefore, a *functional* division of tasks.

We can contrast this with a different type of division of tasks. Marx once described a division of labour he called the 'manufacturing division of labour' (Marx, 1976: 455–491). The manufacturing division of labour sought to reduce the process of production into its simplest and smallest forms, into atomistic movements of body or machine. It is this kind of division of labour upon which Fordism relied, and something more extreme can be found in Taylorism and its 'scientific management' of the labour process. Each part becomes a bit player in something much larger,

itself overseeing a small and incomplete portion of the process. Such a division of labour depends upon a *mechanical* division of tasks, where tasks are grouped together based only upon their relative similarity to one another. The nature of this division is crucial. The mechanical division of tasks groups together tasks that are mechanically similar, but which are individually bit parts in a much larger production process, having little relation to one another. One may, for example, be assigned the task of 'hammerer' whose duty is to hammer in nails across a building site. In a short space of time, such a role would lead one to hammer nails into floorboards, onto the roof, and to hammer a wedge into place. Each of these tasks, while mechanically similar in that each involves hammering, is oriented towards many different functions across the building site. On the other hand, the functional division of tasks assigns to a single space tasks that are united in their function, even whilst each of these tasks bear little similarity to one another. On our building site, for example, one may be assigned the task of building the bedroom, a charge that incorporates many and disparate tasks but each united in their functional orientation.

These two divisions of tasks are radically different with regards to the required 'scope' of coordination. Scope here refers to those with whom one is required to coordinate when performing work of some kind. It includes, principally, those people affected by the changes one intends to make, and therefore those who must be brought into discussions about its implementation and wider effect. Scope is a function both of the organisation of the object and the nature of the division of tasks, both of which determine the degree and rate that changes to the artefact propagate outwards. To take the example of an artefact produced according

to a mechanical division of tasks, we can see that the scope of coordination is very large indeed, where changes quickly propagate to affect the system as a whole. If our hammerer, for example, decided to use glue instead, then the nature of the entire building changes. In this division of tasks, each bit part takes from another bit part its source materials, manipulates them in some way, and passes them on to a subsequent bit part. Production is therefore fundamentally both linear and static in nature, each role is exposed to every other role, and a change in one part has immediate effects throughout the rest of the production process. The manufacturing division of labour therefore provides little scope for localised movement or change within a role before requiring the reorganisation of the production process as a whole.

In contrast, the modularity of Compiz and the free software ecology combined both its functional division of tasks and the elements of the interface and standard to produce a comparatively localised scope. In the first instance, the functional division of tasks created roles that operated in parallel, with each seeing through from start to finish the ongoing work assigned to their functional space. Thus, whilst communication and coordination between modular roles remained necessary, the vast majority of the work of coordination was localised and could remain within the space of the module itself. In the second instance, the elements of the interface and standard made explicit those instances where changes within a module were the proper domain of the module alone or, alternatively, where changes affected elements that were standardised and thus widely expected to behave in a very particular fashion. Our builder dedicated to building the bedroom, therefore, need not coordinate with other builders for the vast majority of tasks required in its construction except, for

example, in the laying of power cables which she knows ahead of time must operate according to certain pre-agreed voltages.

This functional division of tasks coupled with explicit standards for interoperability combined to grant to individual modular spaces a great degree of autonomy. This autonomy could be deployed towards different ends. If we can recall from Chapter Three our discussion of the user-space regime and the domain of the plugins, we can see how this autonomy granted by the machine of the module coupled with the constraints on movement and space of the user-space machine can easily be used to transform the functional division of *tasks* into a functional division of *labour*. That is, to permanently assign one to a modular space. Borrowing from Michel Foucault, we described this as 'cellular individuation,' as a kind of containment. But the functional division of tasks need not equate to a functional division of labour. A single contributor could in the course of their day move from implementing several functions in a variety of programs, granted at each moment the autonomy that each of those modular spaces allows, but free at each moment to move between functional sites. In this, the autonomy of the module coupled with the freedom of movement greatly increased their possibilities of action, which is to say, their freedom. And, indeed, for many contributing to Compiz, its spaces would have formed just one of the many spaces to which they would contribute.

§ Modularisation had a second social effect. The work of creating global order and of coordinating between coders of different modular spaces underwent a process of object fetishisation. Coordination directly between people was diverted, that is,

into both referencing and amending the documents of the standard, where these objects 'stood in' for and almost masked the social nature of this process. Additionally, as the ongoing product of coordinative work between projects, the standards can be characterised as a sedimentation of these direct interactions, and in which they later come to be delegated the task of mediating and ensuring order between projects.

Four days after the Compiz mailing list began, there was an email from David Reveman making the first explicit reference to a standard external to the Compiz project. In reply to another email he wrote,

> Looking at the EMWH spec, I see what I called a virtual desktop, they [the X.Org foundation] call a "Large Desktop". So compiz currently implements one "Large Desktop" […]
>
> (Reveman, David: 2006-04-03 02:57)

The email appears quite trivial, amounting to little more than a correction of terminology. However, the 'correction' of replacing the term 'virtual desktop' with 'large desktop' could equally have worked the opposite way; there was no technical reason for choosing one term over the other. Rather, this was not a correction but rather a *calibration* between the two projects, wherein David chose the EMWH specification as the standard against which to calibrate. One presumes this choice of precedence was because the X.Org project's public EMWH standards both preceded the Compiz project and were widely implemented by other projects.[18] In this brief course of events, the two projects

18. The force or legitimacy of a standard is most closely tied not to the reputation of the body that created the standard, but to the pervasiveness of the adoption of the standard itself. In acknowledging their inability to impose standards,

underwent a minor alignment even whilst contact between the two did not traverse further than the document of the EMWH specification and no direct contact was made with the programmers from the X.Org foundation.

In another instance, a number of strange interactions were being observed as Compiz tried to coordinate with another program known as D-Bus, a program designed to allow communication amongst different programs. Travis Watkins wrote to the mailing list detailing his attempt to discover the source of the bug, concluding that,

> I think the [problem] has something to do with the dbusGetOptionValue being called [improperly] but that change alone doesn't seem to fix it. I've spent about an hour trying to track this one down and am completely lost […]
>
> (Watkins, Travis: 2007-01-01 17:35).

Several emails were subsequently exchanged, each progressively elaborating upon the nature of the bug and putting in place a number of amendments to the code. Finally, David Reveman wrote to the mailing list,

> Hm, after reading some dbus docs I realized that we should always be sending a reply message to method calls unless the no_reply flag is set. I wasn't aware of this… it's fixed now though.
>
> (Reveman, David: 2007-01-02 18:40)

The D-Bus documentation specified the rules governing its interface as well as the allowed and proper set of interactions. As in

---

many standards bodies now talk of writing standards that 'pave the cowpaths,' choosing to instead standardise existing practices. See, for example, the W3C's HTML5 design principles after the failure of XHTML2 (W3C, 2007).

the previous example, it was to this document that David Reveman turned to consult rather than the D-Bus project members themselves, thus enacting the document as mediator for the two projects and their interacting code.

These processes of alignment, calibration, and correction amongst interacting projects were most commonly mediated by the documents of the standards. Queries and other forms of communication that were put directly to external projects were relatively rare and usually prompted one of two responses. In the first instance, when the answer to the query was otherwise available within the documentation, the exchange would generally be characterised as unnecessarily taxing. This scenario was common enough to have its own acronym as a response, 'RTFM,' understood as 'read the fucking manual' (Raymond, 2008). There was therefore a normative compulsion to make use of the mediation of the standards where this was possible. In the second instance, where the query could not be answered by referral to published standards, the existing documentation was cast as inadequate and the exchange prompted amendments, clarification or additions to its content. The standards can thus be seen as a kind of sedimentation of direct interaction between projects, giving permanence to otherwise transient interactions. In the process of this sedimentation of coordinative work both within and between modules, the standards came to be delegated the task of producing order amongst elements at each tier of modular interaction.

The standards were at once a source for global order and simultaneously a target for the changing of that order. In another instance, for example, David Reveman suggested,

> We should try to get the EMWH spec updated sometime soon as being able to communicate a non-rectangular workarea to apps and toolkits is important for the dynamic multi-head support that compiz will be able to do.
>
> (Reveman, David: 2006-11-08 14:16)

Put differently, there did not exist the appropriate interfaces within the X.Org server for Compiz to communicate its emerging functionality, functionality which could be subsequently used by the ecology of programs that were built around the X.Org Server. There occurred in this exchange a diversion over the object of concern. What was initially a concern over the code contained within the X.Org server as well as a number of its associated programs was transformed, without mention, into a concern over amending the document of the EMWH specification. Should agreement have been reached on amending the EMWH specification, these changes would have likely propagated throughout the ecology of programs that adhered to it, including the X.Org Server itself. The manner, however, in which direct coordination was diverted into contestations over an object, in which these contestations 'masked' the desire to change the relationship between numerous projects and their respective bodies of code, demonstrated the mediating work that was performed by the standards, a type of mediation akin to object fetishism.

The inverse operation, where Compiz was on the receiving end of a specification change, also occurred. On 18 April 2006 James Jones, a developer from a related project developing video drivers for the Nvidia chipset, provided advice on how best to conform to an X.Org specification known as 'AIGLX.' At the time, Compiz was not strictly compliant with the AIGLX specification

and, whilst this was not causing any problems, James wanted to implement an option for Compiz that forced it to be strictly compliant with the specification. The AIGLX specification had been written with some foresight as to the future development to the X.Org Server, and strict compliance with its strictures would ensure ongoing compatibility. He reasoned that,

> If, in the future, developers [of the X.Org Server] want to add strict locking as discussed to death on the xorg list, this option could potentially toggle that behaviour as well.
>
> (Jones, James: 2006-04-18 14:22)

In this exchange, we are privy to early stages of a potential change in AIGLX specification which, as James notes, had been the centre of significant debate on the X.Org mailing list. From the point of view of Compiz, however, the arguments and disagreements around this debate were largely localised within the X.Org project and hidden behind the object of the AIGLX specification. In the end, the resolution to these controversies would have been communicated by little more than a humble alteration to a section of the AIGLX specification, part of the ongoing sedimentation of coordinative work, and the history of the debate would have likely been forgotten.[19] The standard would have once again mediated in communicating these changes to the system as whole, ensuring its order even as the system itself changed.

The documents of the various standards were built over time by way of direct coordination amongst affected projects, with each alteration or addition representing the culmination of

19. The outcome of this particular debate remains unknown to this present study.

often difficult and divergent debates. In this sense, these documents formed as a sedimentation of this otherwise fleeting and transient work. For the most part, direct communication across modular boundaries was later rendered unnecessary owing to the mediating work of these documents. These documents 'stood in' for direct contact between projects, mediating to such an extent that the pursuit of changes to how bodies of code and their respective programmers interacted was directed, principally, toward these objects. We can call this a process of object fetishism to the degree that the pursuit of these objects masked the social processes as work.

§The third and final effect of modularisation both within Compiz and amongst its sibling projects was the production of a 'global' order and the emergence of a social structure bearing great similarity to what is known as anarchist federalism. Anarchist federalism was a social structure first proposed in the 19th Century by such early anarchist writers as Pierre Proudhon, Mikhail Bakunin, and Peter Kropotkin, with the specific aim to allow a large mass of people to cooperate and organise their affairs in a manner that ensured power remained dispersed and fully decentralised. Bakunin wrote, for example, 'the future social organisation must be made solely from the bottom upwards, by the free association or federation of workers, firstly in their unions, then in communes, regions, nations and finally in a great federation, international and universal' (Bakunin, 1973: 206). The key features of such a federal structure were to be, firstly, the organisation from the bottom upwards of progressively larger and more encompassing councils, where members would take discus-

sions to councils whose scale was most appropriate for the problem at hand. Secondly, larger and more encompassing councils were merely that: larger and more encompassing. They were not granted authority and could not impose decisions upon their members: larger councils differed only in scale. Decisions at all levels were to be made primarily through consensus or through convincing dissenting members by appeals to the majority interest. Thirdly, where full participation in higher bodies was not possible, lower bodies were to send mandated and recallable delegates to participate on their behalf. These were to be delegates and not representatives, and at no point were they to be granted authority over those who had sent them. Finally, membership and participation within the federal structure was voluntary. The general purpose of anarchist federalism was not policy making and the progressive elaboration of laws, but rather the administration and coordination amongst various groups (Bookchin, 1990: 7).

At a glance, we can note many similarities between anarchist federalism and the free software ecology. It was, like anarchist federalism, organised into progressively more encompassing spaces or, conversely, into spaces of smaller and more specific functionality. Moreover, like anarchist federalism, this organisation was a hierarchy of function, of scale, but it was not a 'hierarchy' in any other sense. We must be careful not to confuse the organisation of the various software artefacts into functional components and subcomponents with a correlating exercise of power. Finally, participation within the various groups and adherence to standards was formally voluntary, thus mandating decision-making models roughly based upon consensus.

The councils and decision-making bodies of anarchist federalism had their parallel, at the highest and most encompassing levels, in the rigorous adherence to, interaction with, and ongoing production of standards documents. There was something of a reversal here, however, when compared to anarchist federalism. In anarchist federalism the identity of the councils was primary, and their agreements and decisions were in a sense their product. That is, it was the council that had continuity, issuing a series of otherwise disparate agreements and decisions over time. In the federalism of free software, however, there were no councils. Rather, the specific standards documents were the focus, around which a group dedicated to its ongoing development came to form, a group that was, in a sense, its product. Here, it was the document that had continuity over time, stabilised by being progressively labelled with higher version numbers. This should remind us of the standards fetishism we encountered earlier. Moreover, at these highest levels, recallable delegation was replaced with direct participation in the production of standards, a feat enabled by the Internet.

Power was highly decentralised and its exercise was roughly evenly distributed throughout the ecology. Participation in producing standards varied from a highly open process in which anyone could participate to that which was entirely closed. This latter situation was, however, rare and placed in jeopardy the likelihood that the standard in question would be accepted and widely adopted. Moreover, none of the standards with which Compiz complied were constructed in such a closed manner. The open production of a standard was more common and those who participated were usually those whom the standard would most directly affect. Adherence to standards was formally voluntary,

though in practice if a standard was widely implemented and a project wished to be compatible or interoperable with other pieces of code, then compliance became necessary. As with Compiz's compliance with the AIGLX specification, total adherence to a standard was not always necessary but, for the sake of future ease, it was often made desirable to be wholly compliant. The opposite was also true: if a standard, in part or in whole, was widely ignored, then it was a standard in name only. These characteristics of the federal structure — being based upon free association, voluntary acceptance of standards, and open participation in standards production — ensured the exercise of power was widely distributed throughout, giving the federal structure an anarchist quality.

For all these similarities with anarchist federalism, however, this was a federalism that did not recognise itself as such. It was most often talked about using the phrase 'community' and also a term I have often used here, 'ecology,' one that resonates strongly with ideas of unplanned order, 'organic' growth, and spontaneity. Instead, its federal nature was an emergent phenomenon that arose out of local desires to coordinate between and amongst different projects, and its primary motor was in the mediating work performed by the documents of the standard. Individually, standards were not constitutive of a far-reaching global order, and their ongoing production was usually aimed towards calibrating projects in a very limited, even local, manner. A single modular space, however, would typically operate under a regime of multiple standards, and work upon any single standard had to take into account other related and perhaps overlapping standards. The overlapping and interrelated nature of standards thus transformed ongoing work upon an individual standard, work

that was otherwise of limited scope, into the constitution of a thoroughgoing global order. That is, global order was a *byproduct*, emerging out of attempts at creating order on a scale considerably smaller in scope. This emergent federalism meant that, unlike the very deliberate and preconceived anarchist federalism, the federal structure of free software did not have an identity, a name, and nor could it represent itself or those it counted as its members: it was primarily a method for working together and only afterwards was it a structure.

§Frederick Brooks sought to bring stability, unity, and 'conceptual integrity' to a large software project by the imposition of a single will, an autocracy headed by a system architect. Only in this manner could the system come together to form a cohesive whole, whose parts understood one another, and where programmers understood their duties and their roles. Only in this manner, Brooks believed, could the tremendous complexity of the project be tamed. But Compiz, and the enormous ecology of software projects of which it was a part, are testament to an entirely different model. The autocracy was replaced with the machine of the Module and its three elements: the module, the interface, and the standard. Modular spaces, in contrast to Brook's autocracy, were granted an internal autonomy in their machinations, bound only by the standards to expose an interface in accordance with its prescription, and facilitated by those very same standards in collaborating with other modular spaces. These standards did not stand outside and apart from the programmers, but were a kind of sedimentation of their ongoing work, the active and ongoing product of their attempts to facilitate order. It was these overlap-

ping and varied standards that ensured the conceptual integrity of the ecology, imposed not from above but generated from the bottom on up, an emergent structure resembling an anarchist federal structure.

Conclusion

We started with the question of utopia and chose to pursue a methodology that focused on those extant practices, the 'what is,' that could also form something of the 'what ought to be.' This approach to formulating something of a utopian vision differs from those that hark back to a golden age of existence or that, alternatively, seek to discern utopia in a future qualitatively distinct from our own. These have their value, no doubt, but this type of 'present tense' utopia perhaps holds greater value for it stresses not rupture but continuity of utopia with certain elements of the present (Gordon, 2009; Newman, 2009). It counts upon the heterogeneity of the world, of an excess that always fails to be captured and subdued by those machines with totalising ambitions.

In pursuing the delineation and subsequent evaluation of each of the machines within Compiz, this is a utopian method that differs from the classical conception of utopia in a number of other ways too. In the first instance, this is a utopian methodology that does not prescribe its own totality to replace the one of today. It is not the prescription of total systems but of disparate

sets of practices, objects and spaces, a largely piecemeal approach to utopia. It is a utopianism that acknowledges the complexity of social life, in that one cannot know ahead of time and in total the good life. One cannot formulate detailed blueprints. This is a utopian methodology that embraces experimentation and the expansion, bit by bit, machine by machine, of a society that is nonetheless radically different to our own. As Paul Goodman once wrote, a 'free society cannot be the substitution of a "new order" for the old order; it is the extension of spheres of free action until they make up most of social life' (Goodman, cited in Suissa, 2009: 247).

If it is a method that embraces the idea of utopia-in-progress it is also one that seeks not its end. This 'open-ended' conception of utopia is an extension of the piecemeal approach; it is an orientation towards ongoing social experimentation and the study and extension of promising subterranean practices. Utopia, to paraphrase Eduardo Galeano, lies forever on the horizon, its purpose being to draw us forward and to imagine differently. In this, the closure, the finality, and the essentially static conception of the classical utopia — those qualities often most troublesome to critics — are rejected.

Finally, this is a conception of utopia that embodies conflict and process. The classical utopia of harmonious coexistence, wherein the forces of opposition and excess are overcome and forever vanquished, is here contrasted with at least one diagram — the Exodus — founded upon an imaginary counter-power, the establishment of a perpetual battlefield in opposition to the emergence of power which is ready, at a moment, to rise forth. The end of history will not be the synthesis and final resolution

of the dialectical forces within society, but rather much like Pierre Clastre's and David Graeber's studies have shown, even egalitarian societies will continue to embody conflict as core to their processes of ordering. Conflict — wholly good and worthwhile — is central to this alternative utopian vision.

§So what of these machines? In this study of Compiz we have discussed two potentially very desirable machines — the Exodus and the Module. We have also come across a third in the ordering mechanism of the Passport that seems quite undesirable, but which nonetheless sheds light on the way in which the control of space and objects can translate directly into control over people. Discussing the desirability of each of the three machines of this study is necessarily a normative manoeuvre. In this briefest of discussions, then, I intend to evaluate each of their ordering mechanisms against what I am calling an 'anarchist ethics.' Anarchism, as political philosophy, is both anti-State and anti-capitalist; it is one that opposes all practices of domination and of representation. In its constructive aspect, it embraces forms of ordering such as economic communism combined with, as we have previously seen, types of social and political organisation such as federalism that ensure the greatest possible distribution of power throughout the social body. In these prescriptions, there is an underlying ethics that principally revolves around a conception of *generalised individual freedom*. This is a conception that is a somewhat messy combination of the ideas of 'freedom from' and 'freedom to,' both of which are not always fully compatible with one another. The former idea is familiar to classical liberal discourse, and includes ideas such as freedom from constraint,

freedom from violence, freedom from fear, and so forth. That is, 'freedom from' is oriented against those oppressive and restrictive operations of power and is conceived primarily as an absence. 'Freedom to' is perhaps a broader conception. If 'freedom from' is familiar to liberal discourse, 'freedom to' is more familiar to socialist discourse. It is the construction of power relations in which individuals are enabled to do things previously impossible. Economic communism, for example, was motivated not simply because it was a more just distribution of wealth, but because in that very distribution the possibilities of life were multiplied. The emphasis on community and mutual aid within anarchism is also derived from such a belief that it is in and through certain types of sociality that we come to enable one another to live lives with a much greater range of possibilities before us.

The desirability of the machine of the Exodus lay in its opposition to centralisation, an opposition that operated through its permanent spectre of desertion. Its elements were everything that made desertion both possible and known, constituting it as a form of imaginary counter-power: a space outside, an absence of restraint and violence, material abundance, and a discourse of counter-power. In the constitution of this imaginary counter-power the machine of the Exodus produced a locus of power that resided in the mass of the people and against an existing order. It was in this locus that the imaginary counter-power would come to wield constitutive effects upon the dominant order. Within Compiz, the machine of the Exodus was instantiated as the forking machine, and its primary constitutive effect was to couple the role of the gatekeeper with the additional role of maintainer. This was the outcome of a kind of exchange, in which the gatekeeper came to be recognised as proper bearer of the code and

thus came to exercise the vicarious powers associated with that role, but was in return burdened with the role of maintainer, a role whose duties were the ongoing articulation of the community of contributors.

The machine of the Exodus operates in at least two different contexts. In the first, as with David Graeber's egalitarian communities, the prospects of desertion operate not against an actually existing power but rather against the prospect of its emergence. It operates in an antagonism against an imaginary lifeworld of witches who threaten to bring the community under their control, and it is precisely in this ongoing antagonism that the machine of the Exodus works to produce the community as egalitarian. In the second context, however, as with Pierre Clastre's Amazonian chiefs and with Compiz itself, the machine of the Exodus works to counteract and limit the powers of an already constituted power. In both cases there is a strong 'freedom from' aspect, as the machine of the Exodus resists and distributes power amongst the body of the community. There is also, in the second element, a limited kind of 'freedom to' in which the community body is enabled to place demands upon constituted power.

The machine of the Exodus embodies something of the right to secede, coupled with the material provisions to make such secession truly possible. One is reminded of the peculiar attitude amongst rural communities during the early months of the Spanish revolution of 1936 in which, for the most part, communities banded together and enacted communistic and cooperative forms of organisation. There were those, however, who resisted such moves, and in most instances they were allowed to go their own way and, additionally, were provisioned land on which to work

with the sole requisite that they did not use the land to reinstate waged relations (Peirats, 1998: 139). The inclusion of the machine of the Exodus as part of a utopian vision embodies a view of utopia as never fully realised, a view of social relations as never fully harmonious, and incorporates within social forms a dynamic element that allows for the regulation of social life against the emergence of centralised power. As with its egalitarian role within Compiz, it seems the machine of the Exodus would find itself well placed as a central and widely instantiated machine in any utopian vision.

The machine of the Module — consisting of the module, the interface, and the standard — transformed the technical artefact of Compiz from a monolithic object into a series of functional spaces, and in doing so provided a method of working on a technical project whilst avoiding centralisation. The spatialisation it created was a very particular and, indeed, a very difficult arrangement of the artefact. The artefact was first broken down into a series of discrete functions with each function then assigned a space of its own. Each space was expected to complete its task within its modular bounds except when part of that task formed the proper function of another module. Crucially, the spatialistion required that the interiority of the each module — that is, the specifics of its implementation — be hidden from without. Communication between modules was therefore managed by the erection of facades known as interfaces, wherein each module's interface presented to the space outside a set of simplified, standardised and stabilised 'hooks.' This modular spatialisation came under the purview of a standard, which specified both the functional decomposition of the artefact and described the interfaces of each module. Modular spatialisation was ostensibly con-

cerned with managing the complexity of the product by breaking it down into a collection of discrete and relatively independent components. But it also and immediately affected the ongoing production of the artefact of Compiz and, indeed, the whole organisation of the ecology of free and open source projects.

The functional spatialisation of the artefact lent itself to a concomitant functional division of tasks. That is, additions or modifications of functionality within the artefact tended only to require changes to a single modular space. Moreover, changes within a module that did not affect the outward behaviour of its interface were essentially invisible to the outside, and thus the scope of coordination with other programmers was oftentimes minimised to the bounds of the module. In this way, the autonomy of labour was greatly increased.

Where coordination across modules was required, this tended to be mediated by the documents of the standards. Being a kind of sedimentation of otherwise fleeting coordinative work, well-formed standards often 'stood in' for direct contact across projects. Moreover, standards were not simply 'read': they also formed the focus for the ongoing development of a modular system, wherein changes to how the system functioned were directed at these documents. From top to bottom, the machine of the Module was instantiated within Compiz as well as in the free and open source ecology generally, and the ongoing reference to and articulation of the plethora of overlapping standards came, quite by chance, to form a kind of decentralised global order akin to anarchist federalism.

The machine of the Module, by itself, does not guarantee the anarchist federalism we observed in the free and open source

ecology. For example, the autonomy of labour that the machine tends to generate can just as easily be used against *labour*, as we saw with the coupling of the user-space machine to produce the cellular individuation of the plugins. In this, the autonomy was transformed into a restrictive isolation, and the functional division of tasks was extended into a functional division of labour. Moreover, it is the manner in which the all-important standards are produced that is foundational to any discussion about 'freedom from' or 'freedom to.' For the machine of the Module can be used, and indeed is used, in thoroughly centralised environments, where standards are imposed and where work is directed from above. In this, we find a kind of hybrid model where the machine of the Module tames the complexity of the task at hand but which is deployed to serve interests apart from workers themselves. The liberatory aspect of the machine of the Module, however, becomes visible when it is coupled with strictly voluntary adherence to standards and, indeed, it was this voluntary nature of free and open source standards that compelled their creation and development to occur in an open, participatory and roughly consensus-based manner.

Even in its best light, however, the machine of the Module represents something of a trade-off between our two conceptions of freedom. As to 'freedom from,' the Module seems to be very much at odds, imposing a number of restrictions namely in the prescriptions of standards which fix the manner in which modular components may relate to one another, and in the establishment of modular borders which restrict functions to certain spaces. Both of these act as constraints upon the manner in which development may proceed. As to 'freedom to,' the Module greatly increases the scope of autonomy, and drastically reduces

the scope of coordination required to embark upon work within a specific part of the artefact. The monolithic nature of the artefact — and of the production process itself — is broken up and thoroughly decentralised.

The monolith versus the module is a good way in which to weigh these constraints and freedoms. The production process around a monolithic object is itself monolithic. The progression of work must be coordinated and calibrated amongst the whole group, and even minor changes must be submitted for approval to ensure breakages do not propagate throughout the object. If decisions are made by a single 'architect' as advocated by Frederick Brooks then such a mode of organisation submits collaborators to the decisions of a single person. Alternatively, if they are democratic, then collaborators find themselves burdened by a collective will. The machine of the Module provides for a type of decision-making that is neither autocratic nor democratic. In this production process, there functions a kind of rough consensus within the limited spaces of the modules, and it is these decisions that make up the bulk of everyday practices. There is a second sphere of decision-making — around the documents of the Standard — that is more formal and more difficult, and which forms a kind of self-selecting consensus, a rule not of the majority but of the 'interested,' of the 'affected.' If the manner in which technical collaboration proceeds is a choice between the monolith and the Module, then it would appear that in the constraints of the Module there lies a relative freedom.

Finally, let us turn to the machine of the Passport. The Passport, if we recall, was founded upon an arrangement of four elements: the border, the port, the files and the document of the

passport. In this arrangement, the borders segmented space and directed movement through the heavily surveilled spaces of the ports. It was at the ports that bodies and objects were intercepted, in which the document of the passport acted at the pineal gland of the machine and linked the body to the interiority of the bureaucratic files. In this interception, knowledge was generated of the body and its movement was subject to the permissions granted in its correlating files. In Compiz this abstract machine was instantiated as the user-space machine and the permissions tables came under the control of a single role known as the gatekeeper. Access to the object Compiz-as-code and to the sub-spaces of the plugins were under the control of this role, and by way of the control of this space a kind of vicarious power was exercised over the community of contributors to the Compiz project.

The Passport is the construction of wide-ranging set of restrictions and controls on movement. In this sense, it represents a violation of the 'freedom from' aspect of anarchist ethics. Moreover, while it does enable the role of gatekeeper, to whom is granted the exercise of an expansive range of powers, this role is confined to but a single person or a small group. That is, this 'freedom to' component is far from generalised. The Passport's instantiation in the vast majority of instances — in the State passport machine, in schools and workplaces, in city centres — thoroughly contravene both the 'freedom from' and 'freedom to' aspects of this ethics, and for which the justifications — which range from fear of the alien, to naked self-interest — are wholly insubstantial. Moreover, the machine of the Passport hardly forms one of those 'minor traditions' or 'subterranean machines' that we could advocate as part of a utopian vision. The Passport, rather than being a machine we should seek to expand, already pervades social

life, and a utopian project motivated by an anarchist ethics should seek, instead, its minimisation.

But whilst uninteresting as part of a utopian vision, the Passport nonetheless sheds light onto one of the principal techniques utilised in the generation of centralised power, namely the control of objects and space. What I have described as the 'vicarious exercise of power,' that is, power which is exercised in and through objects, appears based upon a cursory examination of everyday life to be one of the most prolific, most thoroughgoing, and most mundane techniques of power. It also appears to be one of the techniques most often ignored in social accounts. Whilst the sociological literature is rife with accounts of the ideological basis for centralised power, of winning the consent of the governed and, oddly to a much lesser degree, of the use of violence, coercion and threat, rarely do we see reference to the mundane uses of things in the ongoing performance of centralised power. Michel Foucault's Panopticon remains one of the best examples of a machine that operated in and through the use of objects and the fashioning of space, but even here this machine is all too quickly stripped of its materiality and reduced to a transcendent 'gaze.' The techniques found within the machine of Passport, however, point to the need to include objects and space as foundational to any account of power and, moreover, to any transformation of power as part of a utopian project.

§Free and open source software appears, at first, as a liberatory manifestation. It has as its origins a clear rejection of the commodification of code, and in this socialisation it has given rise to a kind of anarchist communism that exists in the frontier spac-

es of the virtual. In producing one of the most technically impressive artefacts of contemporary times, the ecology of free and open source software stands as an exemplar to the possibilities of collaborative, non-hierarchical relations, and one that stands in contradiction to those who would suggest that it is only in hierarchy, only in economic self-interest and in the sanctity of property that such feats are possible. But this is a hybrid, a mixture of the liberatory and the oppressive. We have uncovered conflicts and battles, ongoing disputes around power and control over spaces, over code, and over status. Moreover, even as it seems in its very existence to contradict those principles upon which the world of Capital is founded, the realm of free and open source software enjoys a strange relationship to that same world. Though beyond the scope of this study, FOSS finds itself not simply being used, but actively contributed to by some of the worlds largest corporations, corporations which have managed to establish for themselves 'business models' around the technical commons and productive output of thousands of programmers.

The intention in studying this hybrid realm has not been to simply advocate the models of organisation used within free and open source software, but to come to understand their different machinations, and to understand in these machinations some of the possibilities of social organisation. The intention here, therefore, has been primarily twofold. In the first instance, it has been an attempt to discern the motors of order within a free and open source software project known as Compiz. In using the concepts of the abstract and concrete machines, we have pieced together thoroughly heterogeneous sets of objects, spaces and bodies which, in their ongoing relation and movement, have effected three primary logics of order: the Passport, the Exodus, and the

Module. In the second instance, the elaboration of these mechanisms of order has been to contribute to a project which extends very much beyond this study, a project that we can call the articulation of a 'utopian mechanics.' Revolutionary movements of the past have traditionally focused their energies on practices of resistance and defence, trusting the shape of the future society to the aspirations of an emancipated working class. But the absence of power does not reveal a genuine lifeworld, it does not set free the true human sociality awaiting release, but rather enacts different power relations. The shaping of the future, that is, cannot be left to essentialist notions of the purity of the revolutionary subject, and if revolutionary movements are to be anything more than a mere changing of the guards they must couple with their resistance efforts a significant constructive project, one that principally involves the articulation, experimentation and spreading of alternative social forms. A utopian mechanics, therefore, seeks to articulate elements of this constructive moment, to elaborate and critique ordering mechanisms and ways of life that not only reject the barbarism of contemporary relations, but which begin to fashion, to borrow from Henri Lefebvre, something of an 'art of living' (Lefebvre, cited in Gardiner, 2000: 78). In accounting for the ordering within Compiz, it is also to this art that I hope this study has made a small contribution.

Emails

Dransfield, Mike (2006-10-06 12:04). *[compiz] Re: [Fwd: Re: compiz coding style].* http://lists.freedesktop.org/archives/compiz/2006-October/000581.html

—— (2007-01-04 07:02). *[compiz] start to develop.* http://lists.freedesktop.org/archives/compiz/2007-January/001173.html

Gandalfn (2006-04-18 12:30). *[compiz] compiz-aiglx patch.* http://lists.freedesktop.org/archives/compiz/2006-April/000106.html

Guthree, Colin (2006-06-19 06:49). *[compiz] Feature request: Multi-head awareness in compiz plugins.* http://lists.freedesktop.org/archives/compiz/2006-June/000284.html

—— (2006-09-23 01:12). *[compiz] Re: gnome-window-decorator -> gtk-window-decorator and some restructuring.* http://lists.freedesktop.org/archives/compiz/2006-September/000454.html

Hearn, Mike (2006-04-01 08:28). *[compiz] [PATCH] Fix a typo in colour loading.* http://lists.freedesktop.org/archives/compiz/2006-April/000006.html

—— (2006-04-06 06:28). *[compiz] g-w-d.c -> my head spins.* http://lists.freedesktop.org/archives/compiz/2006-April/000049.html

Høgsberg, Kristian (2006-09-20 07:27). *[compiz] Fedora Patches.* http://lists.freedesktop.org/archives/compiz/2006-September/000429.html

Hopf, Matthias (2006-06-26 03:35). *[compiz] Patch criterias.* http://lists.freedesktop.org/archives/compiz/2006-June/000297.html

—— (2006-06-28 02:14). *[compiz] Patch criterias.* http://lists.freedesktop.org/archives/compiz/2006-June/000301.html

—— (2006-10-20 05:35). *[compiz] Compiling Compiz.* http://lists.freedesktop.org/archives/compiz/2006-October/000680.html

Jasse, Alex (2006-03-27 12:02). *2 compiz patches.* http://lists.freedesktop.org/archives/compiz/2006-March/000000.html [Referenced only in reply]

Jones, James (2006-04-18 14:22). *[compiz] compiz-aiglx patch.* http://lists.freedesktop.org/archives/compiz/2006-April/000108.html

Krueger, Wulf C. (2006-06-24 17:20). *[compiz] Patch criterias.* http://lists.freedesktop.org/archives/compiz/2006-June/000294.html

—— (2006-06-26 11:21). *[compiz] Patch criterias.* http://lists.freedesktop.org/archives/compiz/2006-June/000299.html

Liebetraut, Thomas (2006-06-25 06:11). *[compiz] Patch criterias.* http://lists.freedesktop.org/archives/compiz/2006-June/000296.html

Müller, Mirco (2006-04-06 05:38). *[compiz] g-w-d.c -> my head spins.* http://lists.freedesktop.org/archives/compiz/2006-April/000048.html

—— (2006-04-06 10:37). *[compiz] g-w-d.c -> my head spins.* http://lists.freedesktop.org/archives/compiz/2006-April/000052.html

—— (2006-04-07 19:06). *[compiz] g-w-d.c -> my head spins.* http://lists.freedesktop.org/archives/compiz/2006-April/000067.html [Referenced only in reply]

—— (2006-04-25 11:28). *[compiz] patch for colored drop shadow.* http://lists.freedesktop.org/archives/compiz/2006-April/000130.html

Reveman, David (2006-03-31 05:28). *Re: 2 compiz patches.* http://lists.freedesktop.org/archives/compiz/2006-March/000000.html

—— (2006-04-01 14:26). *[compiz] [PATCH] Fix a typo in colour loading.* http://lists.freedesktop.org/archives/compiz/2006-April/000011.html

—— (2006-04-03 -2:57). *[compiz] [PATCH] Add option to switcher plugin for current workspace windows only.* http://lists.freedesktop.org/archives/compiz/2006-April/000023.html

—— (2006-04-04 05:32) *[compiz] contributing code.* http://lists.freedesktop.org/archives/compiz/2006-April/000036.html

—— (2006-04-09 10:26) *[compiz] g-w-d.c -> my head spins.* http://lists.freedesktop.org/archives/compiz/2006-April/000068.html

—— (2006-04-10 07:45) *[compiz] g-w-d.c -> my head spins.* http://lists.freedesktop.org/archives/compiz/2006-April/000069.html

—— (2006-04-18 17:51). *[compiz] compiz-aiglx patch.* http://lists.freedesktop.org/archives/compiz/2006-April/000111.html

—— (2006-04-28 04:16). *[compiz] patch for colored drop-shadow.* http://lists.freedesktop.org/archives/compiz/2006-April/000135.html

—— (2006-05-03 03:38). *[compiz] patch for colored drop-shadow.* http://lists.freedesktop.org/archives/compiz/2006-May/000149.html

—— (2006-06-19 06:25). *[compiz] Feature request: Multi-head awareness in compiz and plugins.* http://lists.freedesktop.org/archives/compiz/2006-June/000282.html

—— (2006-09-15 13:13). *[compiz] Re: Discussion about Compiz and working together.* http://lists.freedesktop.org/archives/compiz/2006-September/000424.html

—— (2006-09-27 11:19). *[compiz] plugin templates.* http://lists.freedesktop.org/archives/compiz/2006-September/000477.html

—— (2006-09-28 10:12). *[compiz] beryl fork.* http://lists.freedesktop.org/archives/compiz/2006-September/000487.html

—— (2009-10-04 09:10). *[compiz] Tried out Beryl (Animation pugin).* http://lists.freedesktop.org/archives/compiz/2006-October/000542.html

—— (2006-10-05 13:12). *[compiz] coding style.* http://lists.freedesktop.org/archives/compiz/2006-October/000564.html

—— (2006-10-20 08:47). *[compiz] Compiling Compiz.* http://lists.freedesktop.org/archives/compiz/2006-October/000687.html

—— (2006-11-08 14:16). *[compiz] bug in today's git snapshot with maximizing windows.* http://lists.freedesktop.org/archives/compiz/2006-November/000754.html

—— (2006-11-15 08:26). *[compiz] road map.* http://lists.freedesktop.org/archives/compiz/2006-November/000870.html

—— (2007-01-02 18:40). *[compiz] DBus setting options broken.* http://lists.freedesktop.org/archives/compiz/2007-January/001150.html

—— (2007-02-16 08:06). *[compiz] update on xdevconf07 and beryl situation.* http://lists.freedesktop.org/archives/compiz/2007-February/001413.html

Rosenstand, Mark (2006-05-08 02:37). *Fw: [compiz] patch for colored drop-shadow.* http://lists.freedesktop.org/archives/compiz/2006-May/000159.html

Seguin, Guillaume (2006-06-24 16:24). *[compiz] Patch criterias.* http://lists.freedesktop.org/archives/compiz/2006-June/000293.html

Starr, Shawn (2006-09-15 12:03). *[compiz] Discussion about Compiz and working together.* http://lists.freedesktop.org/archives/compiz/2006-September/000423.html

—— (2006-10-06 12:54). *[compiz] Re: [Fwd: Re: compiz coding style].* http://lists.freedesktop.org/archives/compiz/2006-October/000584.html

Storm, Quinn (2006-05-03 09:39). *Re: [compiz] patch for colored drop-shadow.* http://lists.freedesktop.org/archives/compiz/2006-May/000157.html [Referenced only in reply]

—— (2006-05-03 20:39). *[compiz] patch for colored drop-shadow.* http://lists.freedesktop.org/archives/compiz/2006-May/000151.html

—— (2006-06-24 23:04). *[compiz] Patch criterias.* http://lists.freedesktop.org/archives/compiz/2006-June/000295.html

—— (2006-09-15 15:07). *[compiz] Re: Discussion about Compiz and working together.* http://lists.freedesktop.org/archives/compiz/2006-September/000426.html

Szulecki, Martin (2006-06-19 04:26). *[compiz] Feature request: Multi-head awareness in compiz and plugins.* http://lists.freedesktop.org/archives/compiz/2006-June/000281.html

Watkins, Travis (2007-01-01 17:35). *[compiz] DBus setting options broken.* http://lists.freedesktop.org/archives/compiz/2007-January/001136.html

Bibliography

Agamben, Giorgio (2009). *What is an Apparatus? And other essays.* Stanford, Stanford University Press.

Anderson, Benedict (1991). *Imagined Communities: Reflections on the origins and spread of nationalism.* London & New York, Verso.

Anon. (2000). *The CVS Protocol.* [Online] Available from http://www.wandisco.com/techpubs/cvs-protocol.pdf [Accessed 17 May 2009].

—— (2006). *Software/Compiz.* [Online] http://web.archive.org/web/20060515170917/www.freedesktop.org/wiki/Software/Compiz [Accessed 9 May 2009].

—— (2008a). *Software/CompizTechOverview.* [Online] Available from: http://freedesktop.org/wiki/Software/CompizTechOverview [Accessed 18 December 2009].

—— (2008b). *Development/ExamplePlugins.* [Online] Available from http://wiki.compiz.org/Development/ExamplePlugins [Accessed 18 December 2009].

Arctec Group (2005). *Secure by Design: Security in the Software Development Lifecycle.* [Online] Available from: http://www.arctecgroup.net/pres/tcrugpres.pdf [Accessed 23 April 2010].

Bakunin, Michael (1973). *Michael Bakunin: Selected Writings.* London, Jonathan Cape.

Baldwin, Carliss Y. & Clark, Kim B. (2000). *Design Rules: The power of modularity.* Massachusetts, Massachusetts Institute of Technology.

Barbrook, Richard (1999). *Cyber-Communism: How the Americans are superseding capitalism in cyberspace.* [Online] Available from: http://www.imaginaryfutures.net/cybercommunism_art.pdf [Accessed 21 April 200].

Barlow, John Perry (1996). *A Declaration of the Independence of Cyberspace.* [Online] Available from: https://projects.eff.org/~barlow/Declaration-Final.html [Accessed 3 May 2009].

Bennkler, Yochai (2006). *The Wealth of Networks: How social production transforms markets and freedom.* New Haven & London, Yale University.

Bookchin, Murray (1990). *The Meaning of Confederalism.* [Online] Available from: http://theanarchistlibrary.org/pdfs/a4_imposed/Murray_Bookchin__The_Meaning_of_Confederalism_a4_imposed.pdf [Accessed 22 April 2010].

Brooks, Frederick Phillips (1995). *The Mythical Man-Month: Essays on software engineering.* Reading, Addison-Wesley.

Butler, Judith (1990). *Gender Trouble: Feminism and the subversion of identity.* New York, Routledge.

Castells, Manuel (2000). *End of Millenium. The information age: economy, society and culture.* Oxford, Blackwell.

Chopra, Samir & Dexter, Scott (2008). *Decoding Liberation: The promise of free and open source software.* New York & London, Routledge.

Clastres, Pierre (1989). *Society Against the State.* Trans. Robery Hurley. New York, Zone Books.

—— (1994). Power in primitive societies. Trans. Jeanine Herman. In: *Archeology of Violence.* New York, Semiotext(e).

Colebrook, Claire (2002). *Gilles Deleuze.* New York, Routledge.

Corbet, Jonathan, Kroah-Hartman, Greg & McPherson, Amanda (2009). *Linux Kernel Development: How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It*. [Online] Available from: http://www.linuxfoundation.org/publications/whowrites-linux.pdf [Accessed 26 January 2010].

Crampton, Jeremy W. (2003). *The Political Mapping of Cyberspace.* Chicago, University of Chicago.

Day, Richard J.F. (2005). *Gramsci is Dead: Anarchist currents in the newest social movements.* London, Pluto.

De Certeau, Michel (1988). *The Practice of Everyday Life.* Trans. Steven Rendell. Berkely, University of California.

DeLanda, Manuel (1997). *A Thousand Years of Nonlinear History*. New York, Swerve.

—— (2002). *Intensive Science and Virtual Philosophy.* London & New York, Continuum.

—— (2006). *A New Philosophy of Society: Assemblage theory and social complexity.* London & New York, Continuum.

Deleuze, Gilles (2001). *Pure Immanence.* Trans. Anne Boyman. New York, Zone Books.

—— (2006). *Foucault.* London & New York, Continuum.

Deleuze, Gilles & Guattari, Félix (2004a). *Anti-Oedipus.* London, Continuum.

—— (2004b). *A Thousand Plateaus.* London & new York, Continuum.

Fernández-Medina, Eduardo, Gutiérrez, Carlos, Piattini, Mario & Rosado, David G. (2006). Security patterns and requirements for internet-based applications. *Internet Research*, 16 (5), pp. 519–536.

Foucault, Michel (1995). *Discipline and Punish: The birth of the prison.* New York, Vintage Books.

—— (1998). *The Will to Knowledge: The history of sexuality, volume 1*. London, Penguin Books.

Gardiner, Michael E. (2000). *Critiques of Everyday Life.* Abingdon, Routledge.

GNU Project (1991). *GNU General Public License version 2*. [Online] Available from: http://www.gnu.org/licenses/gpl-2.0.html [Accessed 12 August 2009].

Gordon, Uri (2009). Utopia in contemporary anarchism. In: Laurence Davis & Ruth Kinna (eds.) *Anarchism and Utopianism*. Manchester & New York, Manchester University Press.

Graeber, David (2001). *Toward an Anthropological Theory of Value: The false coin of our own dreams.* New York, Palgrave.

—— (2004). *Fragments of an Anarchist Anthropology.* Chicago, Prickly Paradigm.

Hale-Evans, Ron, McPherson, Amanda & Proffitt, Brian (2008). *Estimating the Total Development Cost of a Linux Distribution*. [Online] Available from: http://www.linuxfoundation.org/publications/estimatinglinux.php [Accessed 28 January 2010].

Harper, Douglas (2001). *Online Etymology Dictionary.* [Online] Available from: http://www.etymonline.com/index.php?term=mediator [Accessed 14 April 2009].

Heritage, John (1984). *Garfinkel and Ethnomethodology.* Oxford, Polity.

Holloway, John (2005). *Change the World Without Taking Power.* Berkeley, University of California.

Jain, Anil K. (2007). Technology: Biometric recognition. *Nature*, 449 (7158), pp. 38–40.

Kim, Nancy S. (2008). The Software Licensing Dilemma. *Brigham Young University Law Review*, 2008 (4), pp. 1103–1164.

Lanzara, Giovan Francesco & Morner, Michèle (2005). Artifacts Rule! How organizing happens in open source software projects.

In: Barbara Czarniawska & Tor Hernes (eds.) *Actor-Network Theory and Organising.* Malmö, Liber AB, pp. 67–90.

Latour, Bruno (2007). *Reassembling the Social: An introduction to Actor-Network-Theory.* Oxford & New York, Oxford.

Lazzarato, Maurizio (1996). Immaterial labour. In: Michael Hardt & Paulo Virno (eds.) *Radical Thought in Italy: A potential politics.* Minneapolis, University of Minnisota.

Lefebvre, Henri (1991). *The Production of Space.* Trans. Donald Nicholson-Smith. Oxford, Blackwell.

Lenin, Vladimir (1917). *The Dual Power.* [Online] Available from: http://www.marxists.org/archive/lenin/works/1917/apr/09.htm [Accessed 17 September 2009].

Levitas, Ruth (2005). *The Imaginary Reconstitution of Society or Why Sociologists Should Take Utopia Seriously.* [Online] Available from: http://www.bristol.ac.uk/sociology/staff/inaugural.doc [Accessed 12 December 2009].

May, Todd (2005). *Gilles Deleuze: An introduction.* Cambridge & New York, Cambridge.

Massey, Doreen (2005). *For Space.* London, Sage.

Marx, Karl (1976). *Capital Volume 1.* London, Penguin.

Mundie, Craig (2001). *Prepared Text of Remarks by Craig Mundie, Microsoft Senior Vice President.* [Online] Available from: http://www.microsoft.com/presspass/exec/craig/05-03sharedsource.mspx [Accessed 12 December 2008].

Narduzzo, Allesandro & Rossi, Alessandro (2003). *Modulairty in Action: GNU/Linux and free/open source software develoment model unleashed.* [Online] Available from: http://ideas.repec.org/p/trt/rockwp/020.html [Accessed 2 February 2009].

Negri, Antonio (2008). *Reflections on Empire.* Trans. Ed Emery. Cambridge, Polity.

Negroponte, Nicholas (1998). *Beyond Digital.* [Online] Available from: http://www.wired.com/wired/archive/6.12/negroponte.html [Accessed 3 May 2009].

Newman, Saul (2009). Anarchism, utopianism and the politics of emancipation. In: Laurence Davis & Ruth Kinna (eds.) *Anarchism and Utopianism.* Manchester & New York, Manchester University Press.

Open Source Initiative (n.d.). *The MIT License.* [Online] Available from: http://www.opensource.org/licenses/mit-license.php [Accessed 22 April 2010].

Peirats, José (1998). *Anarchists in the Spanish Revolution.* London, Freedom Press.

Rawls, Anne Warfield (2002). Editors Introduction. In: Anne Warfield Rawls (ed.) *Ethnomethodology's Program: Working out Durkheim's aphorism.* Lanham, Rowman & Littlefield.

Raymond, Eric Steven (2000). *The Cathedral and the Bazaar.* [Online] Available from: http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ [Accessed 15 November 2009].

—— (2008). *How to Ask Question the Smart Way.* [Online] Available from: http://catb.org/~esr/faqs/smart-questions.html [Accessed 19 January 2010].

Reveman, David (2006a). *Fix up show desktop mode and minimize.* [Online] Available from: http://cgit.freedesktop.org/xorg/app/compiz/commit/?id=9b106375d39ba71c9e56ebcec1d6bb93926179fa [Accessed 7 Aril 2009].

—— (2006b). *Fix typo.* [Online] Available from: http://cgit.freedesktop.org/xorg/app/compiz/commit/?id=930cbbccbb282d05563fb67a8fe9cf654085d8b2 [Accessed 9 April 2009].

—— (2006c). *Add configurable drop-shadows.* [Online] Available from: http://cgit.freedesktop.org/xorg/app/compiz/commit/?id=22436cc83496cc37d7d209692539c13b126a3c76 [Accessed 10 April 2009].

Richie, Dennis (1993). *The Development of the C Language.* [Online] Available from: http://cm.bell-labs.com/cm/cs/who/dmr/chist.html [Accessed 19 April 2010].

Seguin, Guillaume (2006). *Beryl Informations/Announcement.* [Online] Available from: http://web.archive.org/web/20061006234837/forum.beryl-project.org/topic-4591-beryl-informations-announcement [Accessed 25 April 2009].

Suissa, Judith (2009). 'The Space Now Possible': Anarchist education as utopian hope. In: Laurence Davis & Ruth Kinna (eds.) *Anarchism and Utopianism.* Manchester & New York, Manchester University Press.

Shukaitis, Stevphen (2010). An ethnography of nowhere. In: Nathan J. Jun & Shane Wahl (eds.) *New Perspectives on Anarchism.* Lanham, Lexington Books, pp. 303–311.

Torpey, John (2000). *The Invention of the Passport: Surveillance, citizenship and the State.* Cambridge, Cambridge University.

Virno, Paolo (2003). *Virtuosity and Revolution.* [Online] Available from: http://makeworlds.net/node/34 [Accessed 27 August 2009].

W3C (2007). *HTML Design Principles: W3C Working Draft 26 November 2007.* [Online] Available from: http://www.w3.org/TR/html-design-principles/ [Accessed 22 April 2010].

Weber, Max (2004). *The Essential Weber: A reader.* Sam Whimster (ed.). Oxfordshire, Routledge.

Wood, David Murakami (2007). Beyond the Panopticon? Foucault and surveillance studies. In: Jeremy W. Crampton & Stuart Eldon (eds.) *Space, Knowledge and Power: Foucault and geography.* Aldershot, Ashgate.

Woodhouse, Francis (n.d.). *A Simple Compiz Plugin Walkthrough*. [Online] Available from: http://www.downwithnumbers.com/compiz_plugins.html [Accessed 14 January 2010).