In the network: Distributed control in Gentoo Linux

Thomas Østerlie
Department of computer and information science
Norwegian University of Science and Technology
thomas.osterlie@idi.ntnu.no

Abstract

This position paper reports on the findings of an empirical pilot study of Gentoo Linux. Gentoo Linux is an open source Linux distribution developed by a geographically distributed community of volunteers. The reported findings are based on the analysis of a specific episode using actor network theory. With basis in the analysis, it is argued that control in this specific episode can be interpreted as both distributed and local at the same time. Control here being the power to define a problem and make the decision about the appropriate solution to the problem defined. Control, it is argued, is distributed in that it is the function of reciprocal influence among several human and non-human actors. Furthermore, it is argued that control can be interpreted as not inherent in organizational structures or hierarchies, but locally embedded among actors in the decision making process.

1. Introduction

Geographical distribution is one of the distinct characteristics of open source software development. Open source software development has been connected with teams of geographically distributed developers ever since Raymond's first description of the bazaar [1]. Despite the geographical distribution of developers, Raymond describes control in the bazaar as centralized, headed by the 'benign dictator'. Using open source software development as an example of computer-supported distributed work, Moon and Sproull [2] argue that an enabling condition for the success of the Linux kernel are the "capabilities a single leader brings to a project". They argue that the "clear locus of decision-making, singular vision, and consistent voice" are important in controlling this kind of collaborative effort. This supports Raymond's notion of the 'benign dictator'. Control in these two works is therefore understood as centralized.

Mockus and Herbsleb [3] describe the Apache open source web server community in two contradictory ways. On the one hand there is a formal organizational structure for making decisions about code integration. On the other hand, they report that work is not assigned but that individual developers choose what to do themselves. "The choices are constrained, however, by various motivations that are not fully understood." Understanding control as the power to define problems and their appropriate solutions, and thereby making decisions about what tasks to prioritize, Mockus and Herbsleb's description points to a tension between centralized and distributed control.

Picking up on Mockus and Herbsleb's observation, this paper raises the question whether control always is centralized in open source software development? How can we understand the tension between distributed and centralized control?

The paper is organized as follows. Section 2 presents the empirical findings. The section contains a short presentation of the Gentoo Linux case, details of the method employed, and a detailed presentation of the reported episode itself. Section 3 discuses how control can be interpreted in the reported episode. The conclusion draws implications of the discussion, and formulates directions for future work.

2. The case

This section presents the empirical findings. For context, an overview Gentoo Linux is presented first. Then the methods of data collection and analysis that form the basis for this position paper are described. The reported episode is described afterwards, after which the episode is analysed in terms of the mechanics of framing the problem to be solved and what actors take active part in framing the problem.

Gentoo Linux is an open source Linux distribution developed by a geographically distributed community of volunteers. Aiming for advanced users, the distribution is a mix between Linux from scratch and a regular Linux distribution. Gentoo Linux provides the minimum of support for installing a bare bones Linux system. In this way the user can build an installation from the bottom up, tuning it to his exact needs; be it a workstation

installation, a secure server, or a gaming system. That is why Gentoo Linux is also called a meta distribution.

Portage, Gentoo Linux' software distribution system, is the technology that makes this possible. Portage keeps track of the third party software, also called packages, available for Gentoo Linux at any one time. At the time of writing there are over 6000 packages available. Portage also keeps track of which packages have been installed on the local system. Information about installed packages is stored in a database. For each installed package this database contains information such as the absolute path for every files installed by the package, the compiler flags the package was built with, and the package's license.

When installing new packages, Portage compiles the software on the local system. The user can therefore fine-tune such things as compilation flags and additional software support. This information is stored in a set of configuration files.

2.1. Method

The episode reported in this position paper is part of the empirical evidence collected during a pilot study of Gentoo Linux. Data for this pilot study was collected with a number of methods. Archival data was collected from the Gentoo web site at `http://www.gentoo.org`, and from the Gentoo mailing list archives accessed through the `news.gmane.org` service. The IRC logs that form the basis of the analysis which this position paper is based on, were downloaded from Gentoo's home pages. In addition, the pilot investigation involved participatory observations with a software consultant using Gentoo Linux as development platform, and a semi-structured interview with one of the Gentoo Linux developers. The interview was performed according to the guidelines laid down in [4]. Ethnographic field notes [5] were taken in connection during the participatory observation and later written out as a full field report

The episode reported in this position paper is primarily based on the IRC log of the Gentoo managers' meeting from December 15 2003. Using actor network theory, an analysis was performed on basis of the log supplemented by the interview. Actor network theory is a method for analysing the relationship between the technological and the social [6,7]. Unlike traditional software engineering methods that teaches us to categorizes entities into classes such as roles, instances, technical artefacts, organizational artefacts, just to mention a few, actor network theory attributes symmetry to all entities in the network by promoting them to actors. This reflects the basic assumption that all entities in the network are capable of acting upon each other.

Central to actor network analysis is identifying the actors and associations between them. Thinking of actors as nodes and associations as connections between the nodes, the network appears. The network is composed of heterogeneous nodes—technical and non-technical, human and non-human, etc.—that are associated for a period of time. However, the actor network is reducible neither to an actor alone, nor to a network. In addition, the network is seen as constantly shifting, and not as a representation of the original or final state.

In actor network theory the network is an analytical structure constructed by the analyst. Instead of thinking of the actor network as a representation of things out there, it is a conceptual frame, a perspective to interpret social and technological processes. The episode reported in sections 2.2, 2.3, and 2.4 is related as interpreted through the perspective of actor network theory.

2.2. The episode

The Gentoo managers' meeting is a biweekly meeting for Gentoo developers to coordinate activities. The managers' meeting is arranged over the Internet, using IRC. During the Gentoo manager's meeting December 15 2003 [8], the issue of third party utilities operating on Portage's database and configuration files is discussed. Some of these utilities mangle the configuration files, while other utilities no longer work because the Portage database format has changed. One of these utilities, `qpkg`, a utility for querying Portage's database, has accumulated over 20 unresolved bug reports in Gentoo Linux' bug tracking system. The source of all these problems is identified to be code that is out of synchronization with the rest of the system. This kind of problem has been resolved before by introducing the maintainer role. The maintainer is responsible for keeping specific parts of code in synchronization with the rest of the system. The conclusion is that the code in question is outdated because it has not been assigned a maintainer.

An additional response to the problem is to introduce an abstraction layer, an API, on top of Portage's database and configuration files. All utilities accessing the configuration files and database must do so through this API. Two Gentoo developers are assigned to develop and maintain this API.

There is dissent among the participants at the meeting about priorities. Gentoo Linux' chief architect proposes to base the API on Portage's own code. The two developers in charge of the API, while agreeing that this would be a good idea, argue that there are other factors that are more important to take into account when resolving the problem. Especially the issue of missing maintainers for utilities accessing the Portage database and configuration files. The `qpkg` utility is used as an example of these difficulties. The utility was included in the distribution by a developer who later left the project. `qpkg` implements its own code for accessing Portage's database. Responsibility for the utility was handed over to someone

else when the original developer left Gentoo Linux. This second developer went on leave, and `qpkg` was left unmaintained. The problem, while technical in symptoms, is something more and something else. It is also symptomatic for the problems to be addressed by the API developers, in that `qpkg`, like the other utilities, implement its own code for accessing Portage's database and configuration files directly. Without any guarantee for how long the developers for these utilities will stick around Gentoo Linux, the situation that the API is to address is to keep the way utilities access Portage's database and configuration files synchronized even after the original developers leave.

2.3. Framing the problem

The decision to introduce an API on top of Portage's database and configuration files is an answer to a problem the Gentoo developers want to solve. Thinking in terms of actor networks, the problem can in fact be conceptualized as an actor. However, it is not an actor that exists before the meeting starts. It is actually a constructed actor. The problem is "a list of ... trials ... hooked to a name of a thing and to a substance" [7, p.122]. The way the problem is given substance, its framing, is the topic of this section.

In the transcript from the Gentoo manager meeting December 15 2003, one of the developers participating in the meeting states that there are a "slew of util[itie]s lying about". He associates these with mangled Portage configuration files, in that the utilities "hack, slash and mutilate the ... config[uration] files". Then he associates the Portage database with the "util[itie]s lying about", as "these util[itie]s misreads /var/db [the Portage database, author's comment], so as not to be consistent with [P]ortage". Another problem with the "util[itie]s lying about" is that they have overlapping functionality, and none do their tasks particularly well:

"*we don't need five half-working use flag editors. we need one really good one*"

The problem is framed by the developer associating different actors, framing a problem in such a way that the other developers understand it as their problem, too. Figure 1 illustrates how the different actors are associated in framing the problem.

Having framed the problem as a shared problem, its cause is established. The cause of the problem is that the utilities lying about have not been properly updated, as "a few of the existing tools [the same as the utilities lying about, author's comment] don't work with portage 2-0.50 due to API changes [in Portage, author's comment]". That is also why the `qpkg` utility does not work any longer, since there are "20+ bugs [reports] about `qpkg`" that remain unresolved in the bug tracking system. The technical cause of the problem is outdated code, but this is more a representation of the larger problem:

"*now I have 20+ bugs about qpkg assigned to me, it's a mess, and nobody wants to touch it. Who is responsible to maintain it now?*"

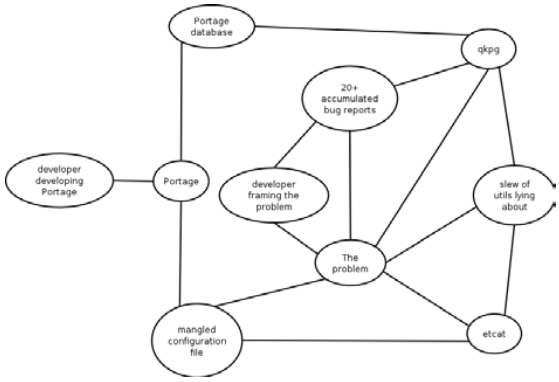

Figure 1 The problem framed

The symptom is that the utilities lying about have not been updated, but this is caused by the fact that there are no one maintaining the "slew of util[itie]s lying about". In this way, the maintainer replaces the problem in the actor network, providing a solution to the situation.

Control is exercised in deciding what activities are to be undertaken, how and when. There are hundreds of unresolved bug reports in Gentoo Linux' bug tracking system. In making the decision about which of these bug reports are to be resolved, decisions about what activities to prioritize are made. Framing the problem can therefore be understood as the power to determine the activities to be undertaken. From this follows that the task of identifying who is in power in the episode above, is the task of identifying who has the power to frame problems.

2.4. Who frames the problem?

At first glance, the problem facing the developers seems to be framed by one of the developers participating in Gentoo manager meeting. As a response to the problem the maintainer role is introduced. The maintainer role, as an actor decoupled from a person, was once constructed to resolve similar situations. In framing the problem at hand in this particular way, the answer to introduce a maintainer becomes a given. Following this line of thinking, one can go as far as saying that the maintainer role participates in shaping the problem. If you have a hammer, all you see are nails. The knowledge among discussion participants that this role exists can be considered constitutive to the problem framing. Looking at the episode this way, the maintainer role is turned from passive to active in framing the problem.

It is highly unlikely that every bug experienced by Gentoo Linux users is reported in the bug tracking system. However, the bugs that are used to frame the problem are those reported in the bug tracking system. Bugs are given priority, severity, status, and assigned to a given person or group of persons for resolution. A bug is resolved when it is fixed or labelled invalid. As long as a bug remains unresolved but assigned to a developer, the bug is a reminder to the assignee. In this sense, bug reports are also active in framing the problem.

Framing the problem is not a function of a single developer or a closed group of developers. Instead, it can be interpreted as the function of a number of actors, both human and non-human. Neither is the power to frame the problem one-sided in that one actor forces other actors to do something they do not want to. Instead, framing is a reciprocal relationship between the Gentoo developers, the maintainer role, and the bug reports.

3. Discussion

This discusses how control can be interpreted in the above episode above. Three aspects of control are discussed. First the implication of the episode in terms of control and organizational hierarchies is discussed. Then we discuss how control can be interpreted as distributed among human and non-human actors. Finally, it is argued that actor network theory makes the interpretation of control as reciprocal among actors likely.

3.1. Relation of control and organizational hierarchy

Gentoo Linux is split into projects and sub-projects. Herds consisting of maintainers are responsible for keeping a set of packages up to date. This is how the Gentoo developers describe their organization in terms of hierarchies and distribution of roles. However, by conceptualizing the way the Gentoo developers talk about the organization during the Gentoo Managers' meeting as an actor network, another view appears. In framing the problem that the API resolves, the maintainer is introduced as an actor in the network. In contrast Gentoo Linux' chief architect does not get through his idea to base the API directly off Portage.

Looking at the organizational hierarchy, the architect is placed farther up than the developer. If control and organizational hierarchies were related, the chief architect would have the power to make his view the prevailing. In the episode above, this does not happen, though. Why not?

Control can be understood as local in the way actors enrol other actors and are enrolled themselves in the immediate actor network. If control was inherent in the hierarchy, the chief-architect should have gotten his view through. That he does not get his view through can be explained by him never enrolling the chief architect role, considered an actor in an actor network analysis, in the immediate actor network.

The implication of the above interpretation is that there need not be an inseparable relation between organizational hierarchy and control. Control can be locally embedded among actors in the immediate network. The actors brought together by the hierarchy have no essential relation to each other, but can instead be understood as dispersed actors temporarily brought together through the hierarchical ordering. By viewing of actors as inherently dispersed, thinking of the organization as an actor network shows that the hierarchical description of organization is just that: a hierarchical description of organization, an abstraction. As such organizational hierarchy need not be inherently connected with control.

3.2. Control is distributed and heterogeneous

In saying that a corrupted configuration file is the same as a missing maintainer, technical (the corrupted configuration file) and organizational (the maintainer) actors are treated as equals. By treating all actors symmetrically this way at the same level of analysis, control can be interpreted as the mutual relationship between heterogeneous actors. Control is not the relationship between action and structures of signification, legitimization and domination [9], but in the direct relationship between actors in the network. A possible implication of this interpretation is that control is no longer purely social, but a function of human and non-human actors, of technological and non-technological actors, of organizational and non-organizational actors. Control becomes orthogonal. It is a function between all actors in the network, regardless of classification schemes. Actors are no longer higher or lower in the organizational hierarchy, technical or non-technical, human or non-human; they are all and the same: actors in the network.

3.3. Control as reciprocal

In saying that control can be understood as local to the immediate network of actors, control becomes both the actors' ability to frame problems, and the ability to limit other actors' framing activities. Control can therefore be understood as more than the traditional control relation within a set of actors

A→B
C→D
D→B
A→E

but as a relationship where actors reciprocally control each other, understood as the relation of

(A, B, C, D, E)

In the latter relationship lies the argument that control is distributed. Control can't be reduced to an actor A's ability to overcome actor B's and thereby exert control over B, as implied in the relationship A→B. It is not one-sided, but distributed. A must not only overcome B's resistance, but the resistance of the other actors in the immediate network. In this sense, in exerting control over B, A exposes itself to the controlling power from the other actors.

4. Conclusion

This paper has argued that traditional notions of control may be inadequate in describing distributed control in Gentoo Linux. Control, it is claimed, need not be limited to the people who seem to be making decisions. Rather, control can be interpreted as distributed among both human and non-human actors. In reported episode, control is distributed among a number of Gentoo developers, the maintainer role, and bug reports. In this sense, control is not distributed in terms of geographical distribution, but distributed as in shared among a handful of human and non-human actors.

While Gentoo Linux is geographically distributed, the interpretation of distributed control is not connected with the geographical distribution. It is, rather, connected with the distribution of elements within an actor network. The key points of distributed control are:

a) that control need not be inherent in the organizational hierarchy, but can be interpreted as embedded in the immediate actor network
b) that control need not be inherent in structures, but can be distributed among actors,
c) that control can't always be reduced to a function of human agency, but may at times be understood as the function of all actors in the network such as tools and organizational roles
d) that control can be a reciprocal relationship between a set of actors

Thinking of distribution this way, similar analysis of distributed control could therefore be equally applicable in geographically co-located software development efforts, too. Distribution is not geographically, but instead understood as distributed among actors.

In arguing that control is distributed in Gentoo Linux, this position paper addresses only the mechanics of control through following the construction of networks through enrolling. The rules of this construction are left untouched. How is it that some actors in the network inscribe stronger behaviour than others? What are the rules for enrolling actors, and what are the rules for excluding actors as valid to be enrolled? These issues need to be addressed in future studies.

The decision to do an API on top of the Portage database and configuration files were only a month and a half old when this pre-study was done. At the time of writing, the API has still to be integrated in a large scale. It is available in Gentoo Linux, but very few utilities actually use the API. A point of future study is to follow up how the implementation of the API and its integration with utilities goes. How is access through the API enforced? How are bugs connected with not using the API handled? What are the effects of introducing the API? Does it lead to lesser problems for utilities integrating with Portage's database and configuration files?

7. References

[1] E.S Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly, Sebastapol, 1999.

[2] J.Y. Moon, and L. Sproull, "Essence of Distributed Work: The Case of the Linux Kernel", First Monday, 5:11, 2000.

[3]A. Mockus, and J.D. Herbsleb, "Why Not Improve Coordination in Distributed Development by Stealing Good Ideas from Open Source?", Proceedings of the 2nd Workshop on Open Source Software Engineering, IEEE, 2002.

[4] S. Kvale, *InterViews: An Introduction to Qualitative Research Interviewing*, SAGE Publications, New York, 1996.

[5] R.M. Emerson, R.I. Fretz, L.L. Shaw, *Writing Ethnographic Fieldnotes*, University of Chicago Press, Chicago, 1995.

[6] M. Callon, "Some elements in a sociology of translation: domestication of the scallops and fishermen of St. Brieuc Bay". *Power, Action and Belief*, Routledge, London, 1986.

[7] B. Latour, "Technology is society made durable", *A Sociology of Monsters. Essays on Power, Technology and Domination*, Routledge, London, 1991.

[8] Gentoo Managers' Meeting Log, `http://www.gentoo.org/proj/en/devrel/manager-meetings/logs/2003/20031215.txt`, last accessed March 1 2004.

[9] A. Giddens, *The Constitution of Society: Outline of the Theory of Structuration*, Polity Press, Cambridge, 1984.