OPEN-SOURCE vs. PROPRIETARY SOFTWARE*

Jean-Michel Dalle
Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson - F-94230 Cachan
Tel: (33) (0)49085995 - Fax: (33) (0)149085996
e-mail: < jmdalle@dir.ens-cachan.fr >

Nicolas Jullien
ENST Bretagne & ICI
Technopole de Brest Iroise - F-29285 Brest Cedex
Tel: (33) (0)298001245 - Fax: (33) (0)298001173
e-mail: < Nicolas.Jullien@enst-bretagne.fr >

** A preliminary version of this article was presented as an invited lecture to the 2001 ESSID Cargèse Summer School. We thank many participants for their comments, and notably Paul A. David, Dominique Foray, Richard Nelson and Edward Steinmuller.*

Abstract:

The article studies technological competition between open-source and proprietary software using a model from interaction theory. We argue that the organizational structure of open-source software, allowed by openness of source codes and by the subsequent development of dedicated communities, is a key feature which, together with compatibility, can allow open-source software to overcome existing proprietary standards. This result depends on the distribution of adopters preferences, but holds when proprietary software producers try to react, even quickly. On the contrary, asymmetric hybridization strategies might successfully allow software producers to protect existing standards.

KEY-WORDS: open-source software, technological competition, interaction theory, standards, hybridization.

1. The case with open-source software: from incentives to technological competition

Open-source software has recently become a major interest both for the software industry and for economic theory. What used to be a Linux[1]-hype has turned into a growing and much-studied phenomenon. Many economic actors now decide for an open-source strategy, i.e. adopt open-source software products and even sometimes publish the sources of the programs they have written instead of keeping them for themselves as used to be the case in the usual *proprietary* model.

In this respect, the main question that has been studied for now in economics is with incentives: why should so many individuals dedicate their work to open-source projects from which they seem to get no reward, while these projects might clearly benefit to many free riders? There are mainly two conflicting explanations here, the first one relying upon some form of inter-individual solidarity, possibly rooted in more or less rational behaviors (Lakhani & Von Hippel, 2000; Harhoff, Henkel & Von Hippel, 2000), while the other is closer to labor economics and, more specifically, to 'career concerns' (Lerner & Tirole, 2000). Although the apparition of an open-source-software economy i.e. of many business firms dedicated to Linux and to other examples of open source software is an important element in this debate (Dalle & Jullien, 2000; Jullien, 2001), as it contributes to create a specific and dedicated labor market, might favor the second explanation, the major problem is that 'career concerns' seem to apply only to a very limited number of *kernel* developers rather than to much more numerous *obscure* developers (Dalle & Jullien, 2001). Though evidence is not clear about how open-source communities are actually organized, notably about the actual proportion of kernel and obscure developers and about their respective roles, and though both categories indeed seem to be essential, the key to open-source software seems to be closer to the face that numerous obscure developers correct infinitely many bugs and therefore critically improve software efficiency (Raymond, 1998).

Anyway, an important consequence of these works is that open-source software is undoubtedly here to stay. But then comes the next and for now neglected question, about technological competition between proprietary and open-source software. Indeed, if open-source software was always to lose against proprietary software, then not only would it have attracted much less attention, but also it would clearly be of considerably diminished interest. As a matter of fact, the interest in open-source software precisely came from the fact that several examples, and notably Linux, had gained significant market shares against proprietary products. Conversely, if open-source software was proven to be able to succeed against proprietary software standards, then it would not only become an even stronger issue for the entire software industry but also an even more interesting subject for economists. Most software technologies are indeed subject to network effects and thus tend to give rise to de facto standards and to monopolies (David, 1985, 1987). If open-source had the potential to durably compete against existing proprietary software standards, then it might become an extremely interesting tool for public policy to restore competition on quasi-monopolistic markets.

Studying this "open-source vs. proprietary software" issue implies the use of a technological competition model with network externalities, which have been

[1] An operating system for computers now seriously challenging Microsoft Windows and others on the server market.

used in economics since Arthur (1989). This paper is precisely dedicated to adapting such a model to the competition between open-source and proprietary software (section 2), and then to inferring several results with both passive (section 3) and reactive (section 4) proprietary software producers.

2. *A model of technological competition between open-source and proprietary software*

Following most recent literature on technological competition, at least since the seminal work of Arthur (1989), we consider that the utility of a technology varies with adoptions, and that adoptions are themselves subject to externalities and network effects. As it has often been emphasized, these externalities can be local and/or global[2]. Local externalities mostly account for compatibility issues and for local word-of-mouth communication of information about technologies with economic agents in the neighborhood. Global externalities account for the improvement of the quality and performance (and price) of the technology during its diffusion. We also account for idiosyncratic components in individual utilities, as potential adopters are clearly heterogeneous: individual costs and benefits associated with the use of a given technology clearly depend on several idiosyncratic factors, and notably here on individual computer skills, a most important characteristic for software technologies.

The utility of a given technology for agent "i" is therefore generally given by:

$$u_{\circ}(i) = k_{\circ}(i) + l_{\circ}(i) + G_{\circ}(i) \qquad (1)$$

where $u_{\circ}(i)$ is the utility of a given software technology for agent i, $G_{\circ}(i)$ (resp., $l_{\circ}(i)$) accounts for global (resp., local) externalities, and $k_{\circ}(i)$ accounts for individual (idiosyncratic) adoption benefits or (individual switching costs.

We further consider that both local and global external economies are simply functions of the local (resp., global) number of adopters of a similar technology. Therefore:

$$u_{\circ}(i) = k_{\circ}(i) + l_{\circ}(x_{\circ}(i)) + G_{\circ}(X_{\circ}) \qquad (2)$$

where $x_{\circ}(i)$ is the number of adopters of the technology in agent i's neighborhood. Note that the last term of equation (2) does not depend any more on each individual adopter but on the global number of adopters of the technology, denoted by $X_{\circ}$.

We consider here an open-source technology, denoted by OS, and a proprietary technology, denoted by P. Therefore:

$$\begin{aligned} u_{OS}(i) &= k_{OS}(i) + l_{OS}(x_{OS}(i)) + G_{OS}(X_{OS}) \\ u_{P}(i) &= k_{P}(i) + l_{P}(x_{P}(i)) + G_{P}(X_{P}) \end{aligned} \qquad (3)$$

Agent i will adopt open-source software iff:

---

[2] See also Dalle (1995, 1997), Dalle & Foray (1998) for other considerations about stochastic interaction models and technological competition.

$$u_{OS}(i) = k_{OS}(i) + l_{OS}(x_{OS}(i)) + G_{OS}(X_{OS}) > u_P(i) = k_P(i) + l_P(x_P(i)) + G_P(X_P) \quad (4)$$

i.e. iff:

$$[l_{OS}(x_{OS}(i)) - l_P(x_P(i))] + [G_{OS}(X_{OS}) - G_P(X_P)] > [k_P(i) - k_{OS}(i)] \quad (5)$$

i.e. if local and global externalities "outweigh" indiosyncratic preferences.

We now turn to some specific characterization of the open-source vs. proprietary software case.

Proposition 1: $$\forall X \; [G_{OS}(X) - G_P(X)] > 0 \quad (6)$$

Proposition 1 expresses the superiority of the open-source development model over a proprietary one, i.e. the superiority of the open-source *organization* when compared to a classical software development organization, as it indeed has early been acknowledged by Microsoft (Valloppillil, 1998).

The 3 main reasons are the following:

(i) Developments in a proprietary organization are mostly ill-targeted because developers are mainly not users, and therefore do not know which functionalities to develop or improve first, or simply where the bugs are. *On the contrary*, open-source communities benefit considerably from a "users as innovators" organization (Von Hippel, 1988), and attract numerous heterogeneous developers which, using their own idiosyncratic experience, correct various bugs and suggest various new developments. *As a consequence*, developments added to open-source software are considerably more efficient for a given level of adoption than for proprietary software.

(ii) Proprietary software producers get incentives to release improved versions only from time to time, so that users are in a way obliged to regularly buy newer versions. Free bug corrections are pretty rare, and usually limited to critical situations: proprietary software producers prefer to wait for improvements to be sufficient to support the release of a new version, i.e. an extra price. *On the contrary*, open-source software is very regularly delivered to users through the release of successive versions which add new functionalities and correct bugs and add minor improvements. *As a consequence*, open-source software is also "continuously" more efficient than proprietary software.

(iii) Finally, the performance of a proprietary technology depends of R&D investments by its producer. These efforts tend to diminish for a monopolist i.e. when network effects drive adoption toward a proprietary standard. More generally, business-firms face a trade-off between investments and profits which has an impact on which share of extra earnings associated with increasing returns of adoption is dedicated to further R&D investments and improvements of their technology. *On the contrary*, open-source communities make no profits, while developers contribute for free. Furthermore, open-source development tends to attract numerous (very) skilled workers which prefer open organizational cultures. *As a consequence*, more developers will generally contribute to a piece of open-source software than to a piece of proprietary software for a given level of adoption.

To summarize, the efficiency of its organization allow open-source software to benefit more from adoption externalities than proprietary software. This

organizational efficiency is directly implied by open-sourceness, which guarantees that modifications and improvements will be redistributed and simply that many developers will have access to the sources and will therefore be able to locate bugs and to develop new features. To put it yet differently, openness of sources is a key condition for open-source software to truly benefit from the creativity of an open-source "community", i.e. from numerous independent software developers.

Proposition 2: $$\forall x \left[l_{OS}(x) - l_P(x) \right] > 0 \qquad (7)$$

A dominant proprietary software producer gets no incentives to make its technology compatible with its open-source competitor. On the contrary, any open-source alternative to an existing proprietary software standard has to be compatible with the existing standard, it has to be possible to use this open-source technology in a network constituted of proprietary technologies. This is typically the case for Linux and most examples of open-source technologies: as for Linux, it is even now possible to emulate Windows on a Linux machine. As a consequence, it is easier to adopt locally a compatible open-source technology rather than a non compatible proprietary technology, for a similar level of local adoption.

Formally:

$$\begin{aligned} l_{OS}(x) &= l_1(x) + l_2(y) \\ l_P(x) &= l_1(x) \end{aligned} \qquad (8)$$

where y denotes the number of proprietary software users in the neighborhood of an agent with x open-source software neighbors. Function l_2 then stands for positive local externalities associated with imperfect compatibility, while function l_1 is considered similar for open-source and proprietary software technologies since we are dealing with similar technologies.

Another line of argumentation for proposition 2 has to do with local informational externalities: open-source users tend to proselytize more than proprietary software users, i.e. to disclose more information, both qualitative and technical, about the open-source software they use. This is mainly due to psycho-sociological reasons, but we wonder whether this reason would not apply as soon as a we would consider any alternative, and notably open-source, to a given dominant standard.

Note finally that neither proposition 1 nor proposition 2 depend on the price of open-source software. As a matter of fact, price considerations would certainly strengthen both, but we believe our results will be stronger if they do not rely on price considerations: namely, the gratuity of open-source software is not necessarily here to stay, if open-source software is here to stay. Furthermore, price is only a limited part of adoption costs. Put differently, we would like to prove that the main economic interest of open-source software is *not* gratuity.

If now we further consider that local and global externalities are simple linear functions, i.e.:

$$\begin{aligned} & l_{OS}\left(x_{OS}(i)\right) = l_{OS} x_{OS}(i) \; ; \; l_P\left(x_P(i)\right) = l_P x_P(i) \\ & G_{OS}\left(X_{OS}\right) = G_{OS} X_{OS} \; ; \; G_P\left(X_P\right) = G_P X_P \end{aligned} \qquad (9)$$

with l_{OS}, l_P, G_{OS} *and* G_P as positive constants, then equation (5) becomes:

$$[l_{OS} x_{OS}(i) - l_P x_P(i)] + [G_{OS} X_{OS} - G_P X_P] > [k_P(i) - k_{OS}(i)] \quad (10)$$

i.e.

$$l_{OS}[x_{OS}(i) - \alpha x_P(i)] + G_{OS}[X_{OS} - \beta X_P] > [k_P(i) - k_{OS}(i)] \quad (11)$$

where of course:

$$\alpha = l_P / l_{OS} > 0 \text{ and } \beta = G_P / G_{OS} > 0 \quad (12)$$

As a consequence of propositions 1 and 2 we have:

$$l_{OS} > l_P \text{ and } G_{OS} > G_P \quad (13)$$

And therefore:

$$\alpha < 1 \text{ and } \beta < 1 \quad (14)$$

Finally, let:

$$a = l_{OS} + G_{OS} \text{ and } b = G_{OS} / l_{OS} + G_{OS} \quad (15)$$

Then (11) becomes:

$$a[(1-b)[x_{OS}(i) - \alpha x_P(i)] + b[X_{OS} - \beta X_P]] > [k_P(i) - k_{OS}(i)] \quad (16)$$

b represents here by definition the relative weight of global versus local externalities (0<b<1) for a given software technology: the closer b will be to 1, the stronger global externalities relative to local ones, and conversely when b is close to 0. a stands for a measure of network effects, both local and global, for this particular technology (a>0): the higher a, the stronger networks effects. a will typically be very high for operating systems, and relatively high for many software technologies[3].

Then, without any restriction since hyperbolic tangent is a bijective function, agent i adopts open-source software iff:

$$th\{a[(1-b)[x_{OS}(i) - \alpha x_P(i)] + b[X_{OS} - \beta X_P]]\} > k'(i) \quad (17)$$

where $-1 < k'(i) < 1$ are distributed according to idiosyncratic preferences of agents in the population. Here, individual preferences are simply compared to externalities and play the role of individual *adoption thresholds*. Note also that the use of an hyperbolic tangent function does not only allows for this normalization of individual adoption thresholds, but also sets boundaries to local and global externalities, which is in line with the idea that the importance of a new adoption diminishes with the number of adopters.

Finally, we will consider here 3 possible distributions of individual preferences (adoption thresholds), namely:

[3] a would be much lower, and b often close to 1, if this model was applied to other, non-software technologies, while propositions 1 and 2 would not hold.

a. A uniform distribution, according to which all preferences are equi-probable. Although this distribution has classically been assessed in many previous works on technological competition with local and global interaction models, it is clearly too restrictive and possibly misleading.

b. A normal, centered (gaussian) distribution: agents on average do not prefer either technology. They are heterogeneous according to a given standard deviation.

c. A bimodal distribution, which will allow us to account for the existence of two subpopulations, as in Arthur (1989), while avoiding to use a double point-mass distribution. This distribution will be assumed to be symmetric, i.e. potential adopters have on average no preference for either technology. But clearly half of them prefer open-source software, while the other half prefers proprietary software. In both cases, theses preferences are distributed in a normal way: such a bimodal distribution is indeed properly given by two (non-centered) normal distributions, each with probability ½.

3. Results with passive proprietary software producer

a. Simulation protocol

For tractability reasons, we classically assume here that neighborhoods are organized according to a 30x30 2-dimensional torus, i.e. each of the 900 potential adopters having 4 relevant local neighbors. We simulate technological trajectories by choosing a random adopter at discrete times according to a given distribution of idiosyncratic preferences[4] and to the existing set of local neighborhoods: we then get his adoption behavior according to formula (17) above, considering also the current global level of adoption. Initial conditions are set to a uniform previous adoption of proprietary software, and we repeat this algorithm up to 500 000 times. We measure the (possibly infinite) time needed for open-source software to become a new standard in the place previously occupied by proprietary software, here defined as a 70% market share. We repeat this entire process for each α and β between 0 and 1 with step 0.1.

Most results exhibit a phase transition: α and β behave as state parameters associated with a sharp discontinuity in diffusion regimes. Above critical values of α and β, diffusion of open-source software is infinitely long – it seems as it will never occur in economic time – whereas below them diffusion is almost sure and fairly rapid. Figure 1 presents typical results of our simulations studies. The large plateau corresponds to an infinitely long diffusion time (i.e. no open-source standard will ever emerge).

[4] For normal and bimodal distributions, we had to deal with negligible distribution tails outside of the boundaries of our model (minus one and plus one): when it happened to be the case, the algorithm simply went to the next step and skipped that particular adopter.

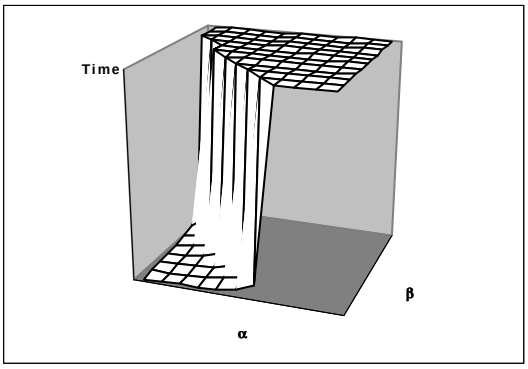

Figure 1: open-source software diffusion time depending on a and b (typical)

Instead of non-obvious 3-dimensional representations, we will present our results here according to 2-dimensional phase diagrams. That is, we construct (α,β) graphs delimiting ranges of values of α and β for which standardization occurs for each software technology. Furthermore, according to our simulation studies, most phase transition frontier between different regimes seem close to linear combinations of α and β and are presented this way.

b. open-source is able to defeat proprietary software

Figure 2a represent the general outcome of technological competition between open-source and proprietary software. For α and β are sufficiently low, open-source software can eventually replace a proprietary software dominant standard[5]. Put differently, the superior organization of open-source software development projects and its compatibility with proprietary software have to be sufficiently high to grant it with a decisive advantage over proprietary standards. Clearly, several characteristics of open-source communities are here relevant in this respect: communities should be large enough and creative enough for β to be low enough. As for α, open-source software should be very well compatible with existing proprietary software. One could even doubt whether this will be enough, and refer then back to what we have named 'proselytism', the kind of which has been created by Microsoft-phobia in favor of Linux and its fellows. Thus a relevant question is here whether open-source software could defeat proprietary software when it was not supported by some kind of psychological feeling favoring the open-source alternative against the dominant proprietary standard, as not all dominant proprietary standard might create as strong an antagonistic response.

[5] These results indeed complement previous results which showed that new technologies had to be "better enough" than existing standards to have a chance to defeat them, or, similarly, that non-linear diffusion and competition processes created endogenous collective thresholds which had to be overcome for a new standard to emerge (Dalle, 1997).

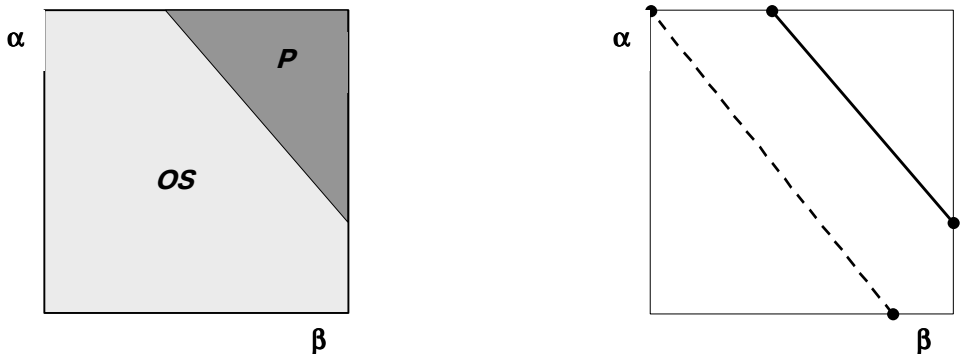

Figure 2a & 2b: phase diagrams, uniform distribution of preferences, b=0.5, a=2.5 (plain line on both figures) and a=5 (dashed line on figure 1b)

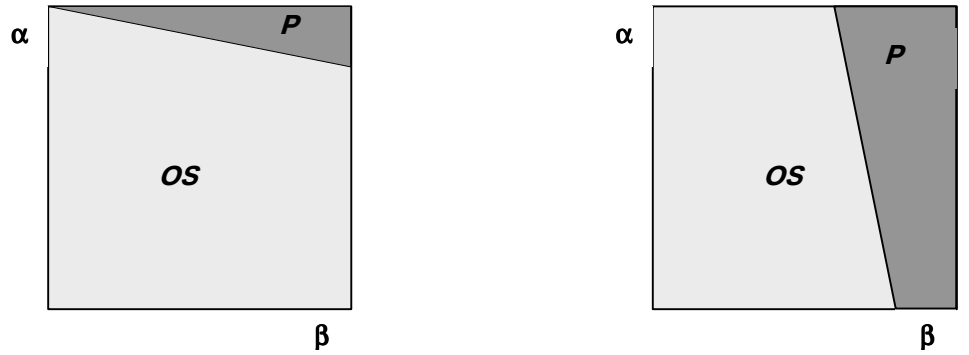

Figure 3a & 3b: phase diagrams, uniform distribution of preferences, a=2.5, b=0.2 (3a) and b=0.8 (3b)

Figure 2b compares this result with results obtained for a technology with stronger network effects (higher a): as expected, α and β should be even lower in this case since network effects first favor the existing standard, which is therefore more difficult to challenge. Fig. 3a and 3b now present phase diagrams when b varies i.e. when local externalities are more influent than global ones (Figure 3a), and less (Figure 3b). Results here are as expected since either α or β play a more important role, except for some asymmetry while equation (17) is symmetric. When local externalities have a strong influence (Figure 3a), open-source software wins almost as soon as $\beta < 1$, while when local externalities have only a limited influence, α should be somewhat lower for open-source software to win. The importance of global superiority when local effects are strong is less than the importance of local superiority when global effects are strong: in all respect, an indirect proof of the importance of local network effects.

c. Results depend strongly on the distribution of adopter preferences

Figure 4 below presents similar results with different distributions of adopter preferences. Clearly here, the uniform distribution is the most favorable situation for open-source software, and also certainly the most unlikely in reality. The least favorable situation is the normal distribution. An interpretation of these results is of course that diffusion is easier when there are more numerous 'early' adopters, i.e. adopters who prefer open-source even when externalities strongly favor proprietary software. Finally, diffusion of open-source software is easier with a bimodal distribution than with the normal one: here again, when there exists a sub-population more favorable to the new open-source software technology, it helps open-source software to be adopted more easily.

Figure 4 also proves that differentiation could be a good strategy for open-source software when it is subject to strong network effects. It is indeed easier for such a new technology to become a standard when adopters mainly belong to two subpopulations which prefer either technology – i.e. when the new technology is able to differentiate from the existing proprietary standard – than when adopters simply normally-distributed heterogeneous preferences. When you want to get rid of a standard more easily (or with less difficulties), then you should first differentiate and typically more specifically target a niche.

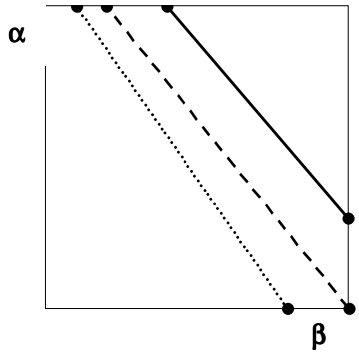

Figure 4: phase diagram, a=2.5, b=0.5, uniform distribution of preferences (plain line), centered normal distribution with 0.4 standard deviation (dotted line) and bimodal distribution with twice 0.2 standard deviation (dashed line)

4. Results with reactive proprietary software producer

a. Simple reactions are likely to be ineffective

Results presented in section 3 above clearly hold only relative to the hypothesis that proprietary software producers do *not* react against the diffusion of their open-source competitor. This is most unlikely, and this is the result why we now turn to study two of two alternative possible reaction strategies.

Let strategy A denote a passive "do-nothing" strategy, and let us first consider the alternative strategy, denoted as B, according to which the proprietary software producer reacts as soon as open-source software reaches x% market share. It can for instance lower its price, although we have already mentioned that price is not necessarily a major determinant of adoption i.e. that demand can be relatively inelastic to price. Another possible reaction by this producer

could correspond to larger investments in R&D to improve the efficiency of its product, be they supported by weaker profits or by debt. Finally, another reaction, considered today by many proprietary software producers, would be to rent their software instead of selling it, so that users would always benefit immediately from bug corrections and new improvements. In all cases, these evolutions represent in our model a significant increase in β, since the utility of the proprietary software technology for users is simply rendered higher for a given level of adoption: therefore, the negative difference with the utility of the open-source software solution is reduced. Typical results are given by Figure 5.

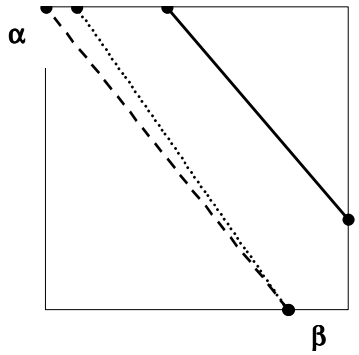

Figure 5: phase diagram, a=2.5, b=0.5; uniform distribution with either strategy A for the proprietary software producer or with strategy B with $\beta \rightarrow \beta+0.2$ as soon as open-source software reaches 5% market share (plain line for both, no difference); centered normal distribution with 0.4 standard deviation with either no reaction (dashed line) or strategy B (dotted line)

Although we consider a quick and important increase in β, we get almost no visible difference between strategy B and strategy A, although we would have expected variations at least in line with the implemented variation of β. Technological competition is here extremely path-dependent: as soon as it has started, it is almost impossible to make it deviate from its trajectory. An interpretation is that initial conditions, i.e. the first adopters of open-source software critically influence its diffusion, even with later a less important advantage in terms of global externalities.

b. Hybridization can be more effective

Let us now consider technological hybridization as another possible strategy for proprietary software producers. By technological hybridization, we mean that either technology can adopt one or more feature of the other. This phenomenon has recently been emphasized by economic historian D.A. Kirsch (1997) during both the early and recent histories of the automobile. Early on, the success of combustion engine vehicles was consistently related to the fact that they had implemented an electric starter, taken from their electric vehicle competitor, which allowed for a much easier start and therefore proved extremely important for many users, and specially for women. Now, there is once again consistent discussions about new “hybrid” vehicles which would possess both a combustion engine *and* an electric engine (or a fuel cell). A new hybrid – variant – might

therefore be on its way, with the idea of combining the environmental interest of electric vehicle and the autonomy and power of combustion engine.

Hybridization in economics works somehow as it does in genetics, where fragments of chromosomes which account for characteristics of the parents get hybridized and mixed together to become the genome of the child: here, technological characteristics from the two technological 'parents' give birth to technological 'children' with several characteristics from each parent: economic hybrids however probably tend to remain closer to either one of their parents. As we have suggested elsewhere (Dalle, Jullien & Simon, 2000), hybridization is a key strategy for producers who face technological competition. It is not only that they try to make their technology better by copying characteristics from other technologies – of course they do –, but hybridization also plays a significantly more important role as influences demand by modifying preferences within different subpopulations of potential adopters who would prefer either technology ab initio, and who might change their minds as the other technology includes several features of their favorite. Modifying the underlying preferences of potential adopters can prove an extremely valuable strategy, specially for a dominant standard to resist technological invaders, as it influences the non-linear dynamics of diffusion and competition processes.

As a matter of fact, we are already observing several examples of proprietary software hybridizing itself with "open-source" features, typically retaining almost all characteristics of proprietary licenses while only its source code becomes accessible. This is why we now implement such a strategy for the producer of the existing proprietary standard. We consider a bimodal distribution of potential adopters: there exist two distinct subpopulations which prefer each available technology. The proprietary software producers then implements features of open-source software during competition. Formally, the distribution of adopter preferences gets modified each time hybridization occurs: the left mode, i.e. adopters who prefer open-source software, moves rightward, i.e. adopters prefer open-source software "less" as proprietary software implements several features of open-source software. The right mode is left unmodified: we consider that adopters who used to prefer proprietary software still prefer proprietary software even once it has implemented several characteristics of open-source software: the new hybrid technology has simply been made even 'better', including new characteristics without losing some therefore with no eviction effect, i.e. with no adopter being 'disappointed' and switching back its preferences to open-source software.

We simulate here two hybridization strategies, one quick and one slow. Namely, strategy C corresponds to an hybridization where the left mode of the distribution of adopters preferences moves to the right of 5% of the distance from its peak and the origin each 1000 periods, while strategy D corresponds to the same move but each 200 periods. Results presented in figure 6 show an important influence of hybridization strategies on the outcome of technological competition between proprietary and open-source software: α and β have to be significantly lower for open-source software to win when the proprietary software producer adopts an hybridization strategy.

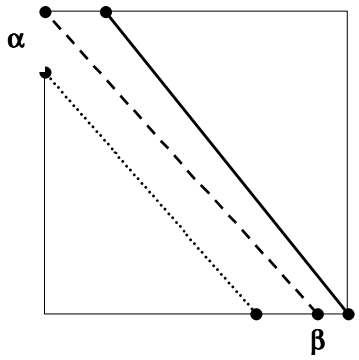

Figure 6: phase diagram, a=2.5, b=0.5: bimodal distribution with either no reaction (strategy A, plain line) or with slow hybridization strategy (strategy C, dashed line) or with rapid hybridization (strategy D, dotted line) from the proprietary software producer

A relevant question here would the sensitivity of users who prefer open-source software to hybridization strategies: they should indeed be credible – recent examples from Microsoft have proven that it might sometimes not be the case – and should also render proprietary software close enough to open-source software to convince users for which the openness of sources is a key condition of their preferences. One could wonder whether it does not purely mean turning proprietary software into open-source software, and if the ongoing invention of less public license will retain enough characteristics of open-source.

5. Conclusion: strengths and weaknesses of open-source software

Open-source software can sometimes defeat proprietary software. However, this is far from being always true. It relies on the efficiency of its organization. Although the basics of this organization are determined by the openness of sources, it relies of course on the community of developers and on its internal organization. In a way, one could interpret the fact that many such communities are not associated with ancillary business firms as a further development of their 'internal' organization. Our models points out that the organizational level is here crucial, since it will crucially determine the outcome of the competition between open-source and proprietary software. In what extent then are open-source software communities able to organize themselves, to 'self-organize', and is it sufficient to grant them with enough competitive power? This is still an open question, which further studies should try to address: this is an important question also if we were to consider public support to open-source projects, as is already the case in Europe, both at national and at the European levels, as a means to correct market failures – monopolies and closed de facto proprietary standards – due to network effects, at least in software markets. Public intervention cannot simply consider existing open-source communities and give them a hint: economists have to suggest ways to help these communities improve their organizational structure.

Apart from this, the competitiveness of open-source software also depends on its compatibility with existing proprietary solutions, and on the distribution of adopter preferences. This can be either a crucial strength or a crucial weakness

for open-source software. To take but one important example, think about the very different destiny of Linux on the server market and on the global PC market: it has won considerable market shares in the first, while it is still stagnant in the second. A straightforward explanation for this, amongst other issues, is the lack of an appropriate graphic user interface (GUI, or desktop) for Linux which is not needed for servers since they are maintained by skilled users – geeks, once again – who even prefer the older 'command line', while normal users need an efficient GUI. In our framework, it means that the distribution of adopter preferences in the server market allowed for the diffusion of Linux, whereas it did not in the global market. A possible conjecture in this last respect might be that there were not enough early adopters, that the left mode of a bimodal distribution was not 'big' enough.

A even more interesting point comes when we add that significant effort is now being done to develop such a GUI for Linux, i.e. this time to hybridize open-source with proprietary software. We have made simulations for this, which have not been reported here as their main conclusion is that asymmetric hybridization by an open-source newcomer has no discernible effect on the outcome of its competition against a dominant proprietary standard. But this absence of result is itself a result here, since it precisely means that developing a GUI for Linux will probably not be sufficient to change its destiny on the PC market. To put it differently, and although these issues are of course still blurred, it might only be part of a potential solution, which should also include a further improvement of Linux compatibility with Windows, and a further increase in the efficiency of its development.

It is then an open question whether such an increase in organizational efficiency is simply feasible, since it certainly depends not only on the number and the personality of kernel developers, but also on the involvement of ancillary business firms which would not only provide dedicated services, as is generally the case with open-source software firms, but which would also act as quasi-editors. But the question then is about how they would then earn money since they could not sell *free* open-source software. Linux will perhaps never replace windows in our offices: perhaps efforts in this direction are even misguided, all the more so as they are diverting efforts from other projects which could win their own open-source vs. proprietary software competition.

6. Bibliography

Arthur W.B. (1989), Competing technologies, increasing returns and lock-in by historical events, Economic Journal 99: 116-131.

Dalle J.-M. (1995), Dynamiques d'adoption, coordination et diversité, Revue Economique, 46: 1081-1098.

Dalle J.-M. (1997), Heterogeneity vs. externalities: a tale of possible technological landscapes, Journal of Evolutionary Economics 7: 395-413.

Dalle J.-M., Foray D. (1998), The innovation vs. standardization dilemma : some insights from stochastic interactions models), in Schweitzer F. & Silverberg G. (eds), Evolution and Self-Organization in Economics, Duncker & Humblot: Berlin, pp. 147-182.

Dalle J.-M., Jullien N. (2000), NT vs. Linux, or some explorations into the economics of Free Software, in Ballot G., Weisbuch G. (eds), Application of simulation to social sciences, Hermès: Paris, pp. 399-416.

Dalle J.-M. & Jullien N. (2001), 'Libre' software: turning fads into institutions, miméo.

Dalle J.-M., Jullien N., Simon B. (2000), The economics of technological hybridization from automobile to Libre software, or why Apple did win against Microsoft, paper presented to the 5th WEHIA Conference, Marseille, June.

David P.A. (1985), Clio and the economics of QWERTY, American Economic Review (Papers and Proceedings) 75: 332-337.

David P.A. (1987), Some new standards for the economics of standardization in the information age, in Dasgupta, Stoneman Eds., Technology policy and economic performance, Cambridge UP, Cambridge, Mass., pp. 206-239.

Harhoff D., Henkel J., Von Hippel E. (2000), profiting for voluntary information spillovers: how users benefit by freely revealing their innovations, mimeo, 26p.

Jullien N. (2001), Impact du Logiciel Libre sur l'Industrie Informatique, PhD thesis, 316 p.

Kirsch D.A. (1997), Technological hybrids and the automobile system: historical considerations and future directions, mimeo, UCLA.

Lakhani K., Von Hippel E. (2000), How Open Source software works : 'free' user-to-user assistance, MIT Sloan School of Management WP #4117.

Lerner J., Tirole J. (2000), The simple economics of Open Source, mimeo, 38p.

Raymond E.S. (1998), The Cathedral and the Bazaar, accessed on 19/04/01 at http://www.tuxedo.org/~esr

Valloppillil V. (1998), Open Source Software, a (new?) development methodology, Microsoft Internal Report.

Von Hippel E. (1988), The sources of innovations, MIT Press.