The Collective Construction of Digital Platforms for Mobile Services: the Mobile OS for Smartphones

Adrien Querbes-Revier, PhD Student

GREThA, Université Bordeaux IV – Montesquieu, France

adrien.querbes-revier@u-bordeaux4.fr

Paper prepared for EMAEE 2011:

7th European Meeting on Applied Evolutionary Economics

"Evolutionary Perspectives on Technical Change and Industrial Dynamics"

February 14-16, 2011

Sant'Anna School of Advanced Studies, Pisa

Abstract: Under different terms – e.g. platform, hierarchic, modular, complex – systems made of interdependent components managed by interfaces have been subject for many studies in the economic and management literature. In this paper, we consider though that these studies are more focus on the components than on the interfaces managing interdependences. Consequently, based on the recent formalizations of the industrial dynamics related to complex products – as "platform architecture" (Gawer, 2009) and "dominant design model" (Murmann and Frenken, 2006), this paper is a very first proposition to formalize our project of interface analysis in these perspectives. The aim is to produce a strong enough set of hypotheses related to interfaces, in order to implement them into computational experiments of system simulations. On the technological side of complex product systems, we argue that interfaces are to be analysed independently from the technological components. Then, on the organisational side, interfaces have counterparts, via the definition of property rights, the active diffusion of technological knowledge and the hierarchical validation. These counterparts are embedded into higher-level standards which co-evolve with technologies and low-level interfaces, according to strategic and economic concerns of the complex product system stakeholders. Finally, we organize this idea according to the representation followed by evolutionary computations, as the first sketch of one simulation model. This paper is a first step in constructing an agent-based model of architectural innovation faced by complex products. Meanwhile, we attempt to keep the intuitive and simple representation of NK models; this is the reason why we have made this step back to theoretical foundations.

1 Introduction

This paper deals the development projects of Operating System (OS) for smartphones. In this extent it aims to question the recent advances in the economic literature on complex products. The point is to produce a robust framework for system simulations – via computational experiments – addressing the issue of *architectural innovation* (Henderson and Clark, 1990) in general, and, more particularly, ongoing in the mobile industry: the reconfiguration of the system linking existing fixed and mobile digital services, thanks to the trivialisation of smartphones. That is why, we are particularly interesting in the representation of the OS as a system, in an economic perspective. In a nutshell, at the technological level, the mobile OS is a *complex system*: a set of components with strong interdependencies, which "*constrains the adaptative potential of systems, and, thereby, the possible paths of evolution*" (Frenken, 2006, p. 3). At the organizational level, it is necessary yet to coordinate complementary and heterogeneous knowledge and skills through collective, localised and historically determined process. However, the strategy of firms – involved in the mobile OS production and innovation – varies from the "open source" community approach to vertical integration. Hence, we aim to explain the rationale and outcomes of this diversity of strategies. In fact, this paper follows a previous paper, which puts forward this diversity from case studies. That is why, we attempt here to build a framework for computational experiments with a "cognitive" perspective (e.g. Marengo and Dosi, 2005), i.e. by articulating complexity at the technological and organizational level.

In this attempt, we survey several trends of the literature in sections 2 and 3. In section 2, we focus on the technological side of complex systems. In line with Simon (1962), one means of reducing complexity is to hierarchically decompose complex systems into subsystems. This approach is to be found in the transversal definition of *platform architecture* made by Baldwin and Woodard (2009), which stems from a decomposition in terms of *modularity* (Baldwin and Clark, 2000), with an emphasis on technological *hierarchy*. It should be recalled that although platform architecture is still a modular system, it is itself split into a *platform* as core, together with its *complements*, interoperability being managed by *interface*[1] specifications. By being transversal, this definition aims to work with many types of complex systems, from product to organization, and, consequently, facilitates the articulation of literature from evolutionary economics with other literature, particularly system engineering. Meanwhile, Murmann and Frenken (2006) have performed a relatively similar effort of concept standardization at the product level. For them, products are a *nested hierarchy of subsystems*, where *core* components inside each subsystem define *dominant design*, by opposition with *peripheral* components. Hence, among other things, they go further in the decomposition of the platform architecture.

According to these definitions, we call – in this section – for the integration of the interface design – and consequently of the product architecture – as a part of the simulation

[1] In this extent, the word *interface* refers to the cement between software components. This differs from *user interface* which refers to the interactions between the software and its (human) user.

framework. In fact, for software projects, modularization arises almost naturally and is defined as a norm of modern software programming (Blume and Apple, 1999; Parnas, 1972). Consequently, interface design is particularly more explicit compared with other products, and is assumed to manage interdependences among modules for the best. For Meyer and Seliger (1998, p. 62), actually : "*in software, the interfaces reign supreme, that is controlling their design and evolution can lead to long-lived systems and is one element of market domination. In fact, the interfaces between subsystems can easily be more important than the subsystems themselves. Microsoft, for example, effectively guides the innovation of thousands of independent software companies by having developed and promoted as a 'standard' the interface mechanisms that allow different programs to communicate with one another in a Web-centric distributed computing environment – known among software developers as 'ActiveX'*". So, most of the surveyed research refers to complex system as modular / nearly-decomposable system; but, it lies on the idea that the few interdependences which remain between modules are detrimental to perfect decomposability of problems, and, therefore, for the organization performance. By endogenizing the complexity of interdependencies as a part of the firm strategy – in line with Baldwin and Clark (2006) – we follow a different approach for the link between technological and industrial dynamics. We aim not to challenge the existing literature: conversely, we consider that this literature has mainly focused on component based innovation, while we are mainly interested by architectural innovation. Thanks to the clarification and standardization effort towards *platform* and *dominant design* concepts, we assume that *interfaces* too could benefit from a better formalisation in innovation models. In this line, we question the ability of various abstractions to do so.

In section 3, we focus on the diversity of organizational strategies in the mobile OS market. Murmann and Frenken (2006) encourage the researchers to analyze the organizational architecture in the light of the complex product architecture. By this way, they refer to research showing the interdependences between technological and organizational structure. For instance, Brusoni *et al.* (2001) explain this interdependence by the level of knowledge specialization facing technological change; in software projects, MacCormack *et al.* (2008) find empirical evidence of the *mirroring hypothesis*, i.e. commercial [resp. open source] software developed by tightly- [resp. loosely-] coupled has a less [resp. more] modular design. But beyond this hypothesis, the interest is to decompose these interdependencies according to the various level of the product hierarchy. Hence, after a brief historical and organisational description of the main mobile OS projects, we analyze the organizational counterparts of interfaces at the subsystem level: i.e. the definition of property rights, the active diffusion of technological knowledge and the hierarchical validation. These counterparts are embedded into higher-level standards which co-evolve with technologies and low-level interfaces, according to strategic and economic concerns of the complex product system stakeholders. We split the standardization processes into two categories, whether they refer to the compatibility standards among layers inside one single platform; or, to the emergence of standards inside one layer.

In section 4, we summarize, from the previous sections, a set of hypothesis to produce the architecture of an economic model for computational experiments. At the current stage of our project, we are not able to produce results because some parts of the model have to be improved to produce a complete and working model. However, we provide three sets of postulates sketching the main parts of the model: the structure representation, the genetic operators (i.e. the rules governing the evolution) and the potential costs and benefits of this evolution. Even if, the results are not available yet, we retain two sets of argument to support this project. In general, computational experiments are interesting because they help to avoid some contingencies of the other empirical methods. For instance, with computational experiments, we are able to deal with complex processes by producing as much data and indexes as necessary. More particularly, it helps to merge data and processes from micro-levels (e.g. interfaces) with meso- or macro-levels (e.g. standardization). Then, some empirical hypotheses lie on results provided via simulations; hence, the interest to merge these hypotheses, in order to test their robustness as a whole. To our knowledge, this work has some equivalent (e.g. Woodard, 2008), but not with a framework based on the evolutionary economics. In this project, we find a major help via the nearly-paradigmatic evolution of computer science and design engineering towards evolutionary computation.

2 Complexity at the Technological Product Level

2.1 Modelling System Interdependencies: the NK Model

In line with Baldwin and Woodard (2009), system interdependencies and system decomposition raise the question of the platform's representation. In this paper though, we extend the idea of representation – as a graphical tool – to the domain of modelling and simulation techniques. They present the Design Structure Matrix[2] (DSM) as a powerful tool to draw a diagram of interdependences inside a platform architecture. This matrix is generally symmetrical and binary, i.e. it shows whether there is a dependency (or not) of the elements in row with elements (of the same platform) in columns. This representation is particularly appealing to link platform architecture and Kaufmann's (1989) NK adaptive landscape model (see Box 1).

This is extensively used to analyze the complexity of innovation process[3], although it originates from biology, more precisely from the evolutionary dynamics of genes. The distance between this approach and the evolutionary dynamics of technologies is very short. For example, the organism may refer to a product made up with technological components (the genes) according to specific techniques (the allelic value). Researchers have also drawn a parallel between this model and project or policy management at the organizational level. These works produced a large amount of interesting results, in the abstract level, for a better

[2] For an application to software, see MacCormack, et al., 2006

[3] See Frenken (2006, chap. 2) who gives a synthesis of the main results for evolutionary economics; see also Ganco and Hoetker (2009) for a survey of NK modelling in strategy literature.

comprehension of the "difficulty" faced by organizations to solve complex problems because of the complexity of interdependencies and / or the search strategies in exploring fitness landscapes.

Box 1: The NK Model

We can split the model down in two parts: the fitness landscape and the search algorithm.

First, the genome of an organism is made up with N *genes*. Each gene has an *allelic* value: 0 or 1, in the model. The set of allelic values at the genome level is the *genotype*; we represent this with a binary string (e.g., 011001, for *N=6*). Each genotype gives a specific *fitness* level for the organism in its environment. Consequently, for the genotype, the set of fitness levels draw the NK *fitness landscape*. In this model, the gene's contribution to fitness depends also on the allelic value of K related genes, named *epistasis*.

This is summarized by:

$$F(x) = \frac{1}{N}\sum_{i=1}^{N} F_i(x_i; x_{i1},..., x_{iK}) \qquad (1)$$

where *F(x)* is the fitness of a given genotype, as an average of the fitness *F(i)* of each gene *i* which depends on the epistasis with the genes $x_{i1},\ldots,x_{iK}$. In fact, one draws randomly the epistatic relations of an organism: for instance, in Figure 1, gene 1 depends on genes 2-6-7-8.

Second, one explores the fitness landscape by mutating the allele of one gene: this means that one draws a new fitness contribution for this gene as well as for the dependent genes. For instance, in Figure 1, if gene 2 mutates, one draws randomly a new value for F_2, F_8 and F_{10} from the uniform distribution *U(0,1)*. One keeps the mutation, provided that it produces a fitness increase and so on and so forth.

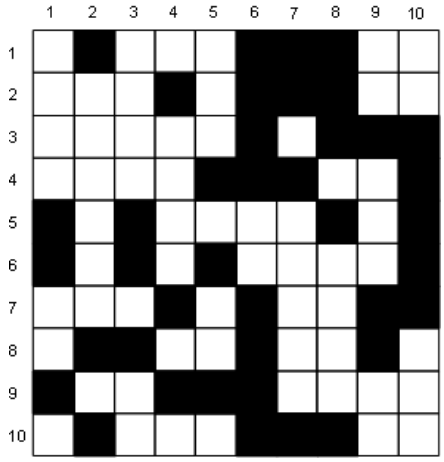

Figure 1: Random epistatic relations for *N=10, K=4*. The genes in row depend on K genes in column.

However, Valente (2008) explains that adaptations have to be made in order to reduce the biological and abstract characteristics of this model. Hence, he proposes a "radical alternative implementation of the core features of NK in such a way to make the model more flexible and adapt for the novel applications it is increasingly put at work". In his model, named *Pseudo-NK*, there are three main differences: (i) interdependencies and allele values are not binary, one can use various level of interdependencies and infinite real-valued alleles; (ii) one can define the location for the global fitness optimum of landscapes; (iii) one can use various search distance for the exploration of landscapes. By doing so, he aims to develop a more realistic approach of evolutionary process driven by human problem solvers with economic incentives. In particular, the interest lies in the possibility to implement this model within wider agent-based models of markets, organizations and so on.

Equally, NK model has major limitations when a modeller tries to address the modularity and decomposability issues. For Frenken (2006, p. 36), modularity and decomposability are not interchangeable concepts: decomposability concept "*holds that a decomposable system is no longer one system, but simply a collection of multiple system of smaller size. […] A modular system is a system that cannot be portioned into subsystems such*

that no interdependencies (epistatic relations) exist between subsystems, but which contains subsystems, called modules, that are mediated by interfaces. These interfaces are elements of a system that connect subsystems such that the only epistatic relations between the subsystems are via the interface standards". Clearly, there is no perfectly decomposable system in mobile platform projects (Yakob and Tell, 2007), but nearly-decomposable (Simon, 1962). Hence, near-decomposability and modularity definitions share the idea that "*a nearly decomposable system – which is the best one could hope for in the real world – is one in which the probabilities of interaction within the subassembly (submatrix) are much higher than those of interaction outside of it*" (Langlois, 1999, p. 4). In fact, dealing with this issue, most of the research using NK models does not generate interdependencies randomly: Figure 1 shows random epistatic relations and, obviously, near-decomposability appears very rarely with this method. Consequently, researchers have constructed manually patterned NK problems, using very abstract hypotheses about the representation of these problems.

In line with Brusoni and Fontana (2011, p. 71), "*advanced technological knowledge about component interactions is used to fully specify and standardize component interfaces and, therefore, to decouple the design of the product architecture (i.e. arrangement of functional elements) from the design of each module*". Consequently, we split these papers into two categories, whether interdependencies between subsystems are one part of the problem or one part of the solution:

- Patterned NK models using given architectures as "problems" does not match with a framework devoted to the simulation of architectural innovation (Frenken, 2006, p. 39); notwithstanding their interest for the other forms of technological change. For instance, several papers study the link between problem decomposability and solver decomposability, according to: e.g. incentives and authority (e.g. Dosi *et al.*, 2003; Rivkin and Siggelkow, 2003); changes in the environment (e.g. Siggelkow and Levinthal, 2003; Brusoni *et al.*, 2004).
- Conversely, some patterned NK models have been designed to analyze the problem architecture. For instance, Rivkin and Siggelkow (2007) use DSMs from "real-world" decision processes to build ten archetypes of epistatic relations. By this way, they show that "*holding fixed the total number of interactions among decisions* [K], *a shift in the pattern of interaction can alter the number of local optima by more than an order of magnitude*" (ibid., p. 1068). Winter *et al.* (2007) use a similar approach, but with higher level of abstraction, by adding a cognitive dimension to the search heuristic. Equally, Frenken *et al.* (1999) analyze the benefit for an organization to match the real structure of a complex problem. Ethiraj and Levinthal (2002, 2004) go further: they test the ability of an organisation to find the true structure of a complex system, according with simple rules, echoing modular operators (Baldwin and Clark, 2000).

Anyway, according to the definitions of platform architecture (Baldwin and Woodard,

2009) and dominant design model (Murmann and Frenken, 2006); these latter propositions are still incomplete. By definition, the architecture of epistatic relations – constructed via NK models – maps elements of the same domain, while interfaces are elements which cannot be reduced to this sort of interdependences. Moreover, the decompositions of this architecture do not take into account the hierarchy between core and peripheral modules.

2.2 Going Back to the Mobile OS: Towards Hierarchical Decomposition

In order to produce such a representation Murmann and Frenken (2006) refers to a model generalizing Kaufmann's (1989) NK model, described by Altenberg (1994). Both models share the same representation of genome and genotype. However, the complexity (related to epistasis in Kaufmann's model) is now related to *pleiotropy*, i.e. the number of functions of the organism affected by one gene. The set of genome's pleiotropy is a map named *genotype-phenotype map* (see Figure 2), since it connects genes and functions of the organism. The K parameter disappears: the only condition is that each function must be connected with one gene at least. Meanwhile, the number of genes affecting one function is named *polygeny*. Consequently, in the product perspective, while with Kaufmann's NK model one was looking at the interdependences between technological components (the genes), one represents now the interdependencies between technological components and the "service characteristics" (the functions) as seen by end-users and / or project managers.

Hence, there are two domains of interdependencies: between technological components as well as between technological components and related characteristics. This notion comes from Saviotti and Metcalfe (1984), based on the idea of Lancaster (1966) to define products as a set of characteristics. In their view, the technical characteristics are the "internal" characteristics, i.e. the components. Their combination produces "external" characteristics, i.e. the service as seen by end-users. For hierarchical decomposition, the technical components with high pleiotropy are core components because they affects many services, and reciprocally. So, Murmann and Frenken (2006) split high pleiotropic components into core components and interfaces: at each level, interfaces integrate components into subsystems or subsystems into systems (see Figure 2).

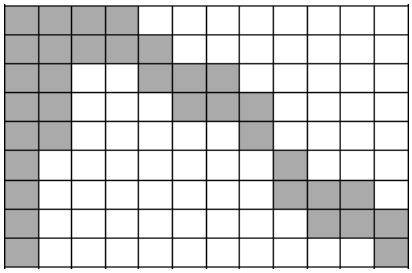

Figure 2: "Example of a genotype–phenotype map with 9 services characteristics (rows) and 12 technical characteristics (columns). Column 1 is an interface standard with a pleiotropy of 9. Column 2 is a core component with a pleiotropy of 5. All other components are peripheral with a pleiotropy of 2" (Murmann and Frenken, 2006, p. 941).

As an *abstraction*, this mapping has interesting features. Genotype-phenotype maps

connect objects from two different domains: this abstraction is shared by other approaches. For instance, Danilovic and Browning (2007) name these cross-domain maps Domain Mapping Matrices (DMM) and show their relative advantages for complex product development projects, compared with DSM. Equally, Baldwin and Woodard (2009) introduce *layer maps* as a two-domain representation of the industry competition: they show how several firms compete on several layers of complementary components from the platform architecture. This is a relatively intuitive way to represent the industrial structure in terms of vertical and horizontal integration.

More particularly, for Antonelli (2005, p. 55): "*knowledge emerges out of the inductive process of abstraction and generalization rather than from the deductive process of application of general ideas to specific circumstances*". Hence, these maps are particularly interesting because they merge technological representations and the effect of technologies on product's services / functionalities. This second domain is highly dependent on the own vision of its modeller, and, therefore, provide a strong connection between technology and organization. Moreover, this kind of abstraction exists also in computer science, where operating system is represented via layers. Meanwhile, by using abstractions, "*empirical research on dominant designs requires a judgment about whether two designs are different or the same. The outcome of these judgments depends crucially on the level of resolution or granularity one brings to the analysis*" (Murmann and Frenken, 2006, p. 934). With high-level abstraction, the OS is nothing more than an interface between applications (software) and physical resources (hardware). This representation "*can reduce the performance, increase the complexity, and limit the functionality of application programs*" (Engler *et al.*, 1995, p. 2); hence, the interest for lower-level abstraction. Equally, we have to define the modular system boundaries: since the OS is made up of digital modules, our criterion is to define the modular system as technologically homogeneous, therefore made up of the OS and its digital complements, i.e. the digital modules supporting the related applications. Figure 3 shows the Android architecture representation. Put in our framework words, we see the services (applications) at the top, resulting from core and peripheral digital modules, organized hierarchically from the kernel to the application framework.

This abstraction does not put forward the interfaces, they are actually implicit. To know where the interfaces are, we have to go further into the abstraction, via the concept of *code reusability*. In fact, "*in software product lines, there are two kinds of assets, core assets and custom assets. A core asset contains a set of domain specific but application independent components that can be adapted and reused in various related applications. A custom asset contains a set of application specific components. In software product line, core assets, which are designed for reuse within the specific domain, are more stable than custom assets, which are designed for a specific application*" (Yu and Ramaswamy, 2006, p. 5). This idea is consistent with Murmann and Frenken (2006)'s proposition as well as with Baldwin and Woodard (2009, p. 25): "*a platform architecture displays a special type of modularity, in which a product or system is split into a set of components with low variety and high*

reusability, and another set with high variety and low reusability", respectively the core and complements. More particularly, for OS, the major core asset is the kernel (see Figure 3).

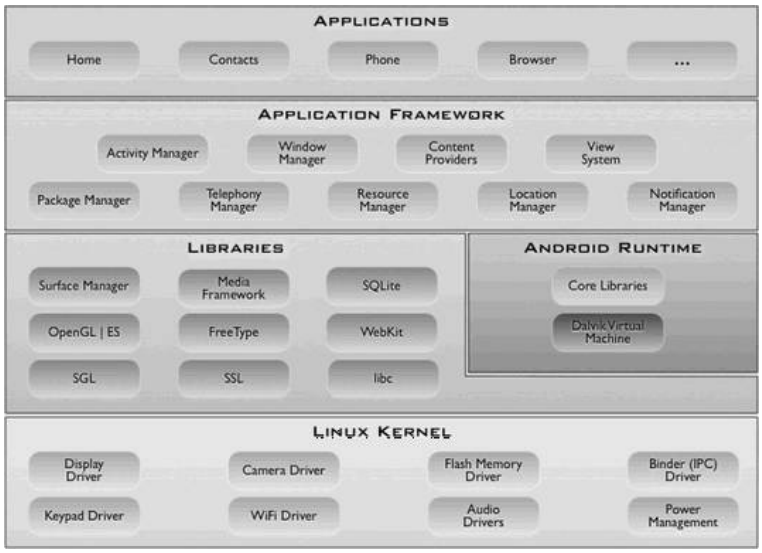

Figure 3: Android platform architecture

Source : http://developer.android.com/guide/basics/what-is-android.html

Consequently, Luer *et al.* (2001) shows that the reusability of software components exhibits different types of modularity, via different types of code reuse[4]. There are three possibilities:

- *Copy-and-paste*: pieces of code are directly reused in other software. Eventually, the copied code will not be recognizable, and, therefore, could be changed and evolves in different direction.
- *Statically linked library*: pieces of reused code are identifiable as modules via libraries. But this library are pasted in all the executable files where they are necessary, "*each of these has to be rebuilt when the library is updated*".
- *Dynamic linking*: this piece of code exists in a single copy (as a single module) and is accessible for programs which need it. There is no change to do on these programs if the module is updated, but this increases complexity by introducing dependencies.

Except for the first case, these "links" define the interdependences between modules: they are the interfaces. First, they describe the architecture, since "*we have a hierarchical structure if a certain relation may be defined between the modules or programs and that relation is a partial ordering. The relation we are concerned with is 'uses' or 'depends upon'*" (Parnas, 1972, p. 425). Second, in computing science, they are explicit, because they provide all the information necessary to interact with the code which could be, therefore "hidden". One names them Application Programming Interface (API); they define the rules,

[4] "*Code reuse means that an application reuses a component by accessing its actual code (whether in source or in compiled form), loading it into memory, and then executing it*" (*ibid.*, p. 2).

the vocabulary and the functionalities to connect programs from various layers. Consequently, we assume a small change, compared with the definition of Baldwin and Clark (2000, p. 77), but coherent with the definition of Baldwin and Woodard (2009): the interfaces, as interdependences among components, define the architecture or design; they are the design rules.

2.3 The Evolutionary Processes

The biological analogy is not limited to the representation of complexity: it is also used to describe the system evolution. For Luer *et al.* (2001, p. 1), "*evolvability is the property of programs that can easily be updated to fulfil new requirements; software that is evolvable will cost less to maintain. A component-based application is evolvable if it is easily possible to exchange individual components without changing others*". It echoes clearly the idea that "*modularity would enhance the ability of the genetic system to generate adaptive variants, which one can refer to as its 'evolvability'*" (Altenberg, 2005, p. 2) in biology, as well as the idea of *flexibility* in economics and management (Sanchez, 1995; Baldwin and Clark, 2000). But, concretely, how is performed this evolution? In NK models, evolution consists in allelic mutation of genes in order to increase fitness. This vision is simplistic for biological organism as well as for software projects.

That is why Altenberg (1994) proposes the method of *constructional selection*. Hence, based on a genotype-phenotype map, one fills and improves step-by-step an empty genome, regarding the algorithm shown on Figure 4. So, the map is not randomly constructed *ex ante* with given values of N and K, but it is randomly constructed *in vivo* according to its shape at the previous step. The landscapes generated in this way differ radically from usual NK approach (see Figure 5). This double process of adding and mutating genes has not as main objective to mimic the genome growth of an organism, but rather to back theoretical hypotheses. Hence, constructional selection encourages modularity, since the pleiotropy of new genes decreases step after step: the latest genes are only connected with the very few functions which have not reached their maximum fitness value. By this way, applied to industrial dynamics, this model proposes an abstraction of temporality. Murmann and Frenken (2006) apply this representation to product architecture, putting forward that "*once a design has settled on particular variants of core components, further advances are concentrated in peripheral components only*" (*ibid*., p. 941). Applied to software development, Von Krogh *et al.* (2009) find opposite results (peripheral components are introduced earlier and core components are subject to evenly changes), while MacCormack and Verganti (2003) show that, facing technological and market uncertainty, software projects practitioners who achieve the best performances are those who invest in architectural design during the early stages of the project. For us, these differences in the observations come from various representation of the temporal organization of software projects. A linear sequence may be observed, since "*a modular design process has three basic stages: (1) the formulation of design rules; (2) parallel work on hidden modules; and (3) testing and integration*" (Baldwin and Clark, 2000,

p. 246). However, this sequence is cyclical because the interface architecture is to be continuously improved as long as new interdependences among components are discovered or created. Unlike biological organism, texting and integration of software system does not mean the death of non-fitting projects, but a new cycle of redesign, debugging and so on. For instance, Narduzzo and Rossi (2003) study a set of open source projects and observe that architecture of software are inherited from previous software system to avoid the problems related to designing the architecture from scratch. This method is relevant for our case, since we have seen that one mobile OS platform (Android) reuses the kernel of Linux (Figure 3, above). Equally, Yakob and Tell (2007) study one complex mobile platform project, where a basic architecture is designed according to an initial set of functionalities and benefits of cyclical improvements while new sets of functionalities are added.

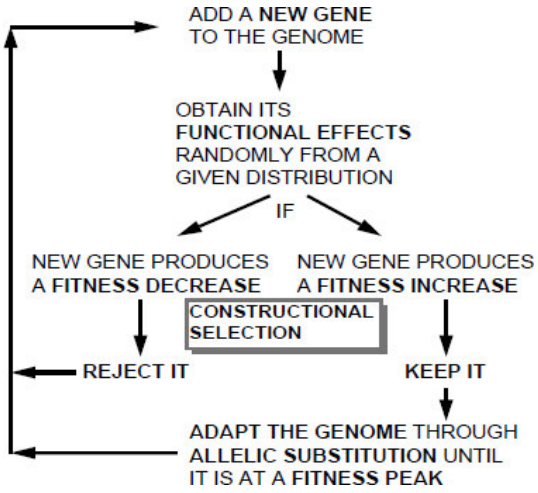

Figure 4: The genome growth algorithm with constructional selection (Source: Altenberg, 1994)

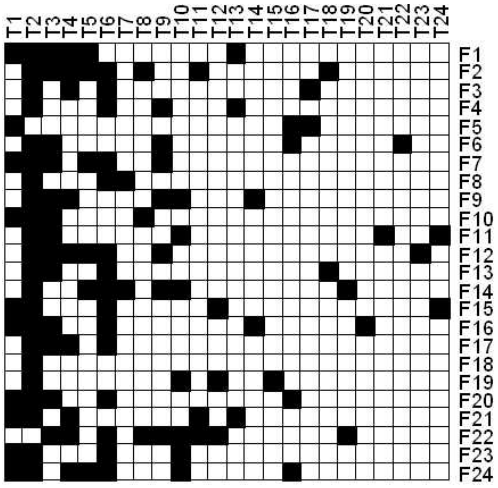

Figure 5: Random genotype-phenotype map with constructional selection

This process could involve *speciation* (Frenken, 2006, p. 46). This idea refers to a bifurcation of a system into several trajectories adapted to specific selection environment, because of end-users heterogeneity. This situation is not detrimental, compared with standardization, if a single platform core is not able to serve the whole set of service characteristics asked by end-users. However, software platforms are more flexible than physical platforms and biological organisms, since they are less constrained by physical laws: so, speciation is less likely to occur. For instance, in the case of a major mobile OS platform (Android): "*updates to the framework API are designed so that the new API remains compatible with earlier versions of the API. That is, most changes in the API are additive and introduce new or replacement functionality. As parts of the API are upgraded, the older replaced parts are deprecated but are not removed, so that existing applications can still use them.* […] *You can determine the lowest possible platform version by compiling the application against successively lower build targets. After you determine the lowest version, you should create an AVD* [Android Virtual Device] *using the corresponding platform*

version (and API Level) and fully test your application"[5]. In a purely technological perspective, we deduce, therefore, that mobile OS resists bifurcation thanks to interfaces. The set of API mimic the platform evolution when this evolution refers to replacing or adding functionalities to the platform. However, in the emerging market of mobile platforms, we observe a relative *fragmentation*, i.e. competing platforms which attempts to provide the same services via competing and incompatible technologies.

3 Complexity at the Organizational Level

3.1 The Diversity of Organizations Supporting Mobile OS Projects

Table 1 gives an overview of the mobile OS market. The decreasing leader – Symbian – appeared in 1998, from a former PDA OS (EPOC, the OS of Psion's handheld devices), via Symbian Ltd, a firm shared by mobile handset manufacturers (Ericsson, Motorola, Nokia and Psion). from 2nd December 2008 to 8th November 2010, when Symbian had been released under open source specifications and collective innovation, via the Symbian Foundation – which creation was announced on 24th June 2008, by Nokia, together with AT&T, LG Electronics, Motorola, NTT DoCoMo, Samsung, Sony Ericsson, ST-NXP Wireless, Texas Instruments and Vodafone. But, now Symbian is managed in-house by Nokia under its complete supervision. The BlackBerry OS is provided by Research In Motion (RIM), a firm which provides both the BlackBerry smartphone and its OS. RIM has a long experience in connected mobile devices, since it provided in 1995 the Inter@ctive Pager, capable of sending and receiving text messages through a specific wireless network, Mobitex. Apple launched its PDA, the Newton, in 1993, but exited the market in 1998, without any real commercial success. At the same time, Microsoft has been active on the PDA via the Pocket PC and its OS (Windows CE), since 1996. Microsoft never left the mobile OS market, continuously improving its Windows CE, known as Window Mobile, since 2003. Apple returned to this market only in 2007, launching the iPhone and its OS (iOS). Comparatively, Android is a fast growing new entrant, since it is only on 5th November 2007, that thirty-four companies (including Google) announced their involvement in the development of an "*open platform for mobile devices*"[6] – Android – within a new inter-firm organization, the Open Handset Alliance (OHA).

The mobile OS – compared to the desktop computer OS – is particular, because it needs better integration with both the device and the end-user service (Funk, 2001) to give the easiest access to the embedded mobile services (Kelly, 2006). Consequently, mobile OS are a major concern for *systems integrators*, i.e. "*companies that rely on wide and dispersed networks of suppliers of specialised components and capabilities, yet maintain broad and deep in-house capabilities*" (Brusoni *et al.*, 2004, p. 5). These system integrators come from

[5]Source : http://developer.android.com/guide/appendix/api-levels.html
[6] Source: Open Handset Alliance, *Press Releases*, November 5, 2007.

different sectors, follow various strategies, and have to coordinate the innovations from heterogeneous technological fields (Kelly, 2006; Maula *et al.*, 2006). Knowing that the degree of product decomposability depends on the existing state of knowledge (Buenstorff, 2005).

	2008		2009		2010 Q1		2010 Q2		2010 Q3	
Operating System	units	%	units	%	units	%	units	%	*units*	%
Symbian	72 933.5	**52.4**	80 878.6	**46.9**	24 069.8	**44.3**	25,386.8	**41.2**	29.480.1	**36.6**
BlackBerry OS	23 149.0	**16.6**	34 346.6	**19.9**	10 552.6	**19.4**	11,228.8	**18.2**	11 908.3	**14.8**
Android	640.5	**0.5**	6 798.4	**3.9**	5 214.7	**9.6**	10,606.1	**17.2**	20 500.0	**25.5**
iOS (iPhone)	11 417.5	**8.2**	24 889.8	**14.4**	8 359.7	**15.4**	8,743.0	**14.2**	13 484.4	**16.7**
Windows Mobile	16 498.1	**11.8**	15 027.6	**8.7**	3 706.0	**6.8**	3,096.4	**5.0**	2 247.9	**2.8**
Linux	10 662.4	**7.6**	8 123.5	**4.7**	1 993.9	**3.7**	1,503.1	**2.4**	1 697.1	**2.1**
Other OSs	4 026.9	**2.9**	2 305.6	**1.3**	404.8	**0.7**	1,084.8	**1.8**	1 214.8	**1.5**
Total	**139 287.9**	**100.0**	**172 373.1**	**100.0**	**54 301.4**	**100.0**	**61 649.1**	**100.0**	**80 532.6**	**100.0**

Table 1: Worldwide Smartphone Sales to End Users by Operating System (Thousands of Units)[7]

Clearly, the mobile OS sponsors, as system integrators, have not the same positions in the system hierarchy. For Apple and RIM, the system integration covers all the subsystems: the OS is developed in-house, respectively iOS and Blackberry OS; it is not available for competing handset manufacturers; third-party applications need validation to access their online application stores, respectively the App Store and App World; even some wireless interfaces are imposed to network operators. Microsoft and Nokia, comparatively, perform sub-level integration. Microsoft does not manufacture handsets, while Symbian OS was developed by many handset manufacturers during the Symbian Ltd and Foundation era and, therefore, shipped on their devices. They have also validation tools for third-party applications on the Marketplace and OVI. Last but not least, Google perform an indirect integration of the Android system. In fact, while the Alliance supports the diffusion and enhancement of Android, the technical directions and third-party validations are given directly by the Android Open Source Project (AOSP), managed by Google employees only. So, the AOSP "*welcome all uses of the Android source code, but only Android compatible devices - as defined and tested by the Android Compatibility Program – may participate in the Android ecosystem*"[8].

3.2 The Organization in the Subsystem Perspective

This description of the system integration with a high-level of abstraction (handset – OS – application) put forward the necessity of complementary concepts to go further into the

[7] Source: Gartner, *Press releases*.
[8] http://source.android.com/faqs.html

lower-levels of abstraction and, therefore, into the subsystems.

We see that mobile OS sponsors are not the only stakeholders of the system. In our view, this refers to the legal ownership of the components – i.e. the intellectual property – as a tool to discriminate the knowledge accessibility. The *proprietary* versus *open* source strategies define the scope for digital platforms, but modularity encourages more subtle strategies, via partial licensing. For Bonaccorsi *et al* (2006, p. 1094), software firms tend to hybridize open and proprietary strategy rather than follow a pure model. This idea is consistent with West (2003, p. 1279): proprietary platform vendors' strategy evolves from fully proprietary to open standards and, then, to open source, but the strategy is always hybrid ("opening parts", "partly open"). A certain number of reasons encourage these sponsors to operate this transition: e.g., the market environment (ibid., 2003), developers' preference (Sen *et al.*, 2008), users' involvement in innovation (von Hippel and von Krogh, 2003), project characteristics (Lerner and Tirole, 2005). At the same time, the influence of intellectual property varies with its position in the platform architecture. For instance, openness at the platform (core) level provides a good picture of *co-opetition* (Nalebuff and Brandenburger, 1997): cooperation on platform standards, competition on complements. These strategies are possible thanks to "hybrid" open source licenses, for commercially-based contributions. We can simplify this question via the concept of *copyleft*[9]. In line with Pénin (2008), there is both *strong* and *weak openness* (copyleft, here) depending on whether the stakeholders have to ask permission, or not, to use the code, even if the source code is open. Consequently, the definition of the mobile as open or proprietary must be done carefully: the trade-off between open and proprietary licensing is largely at the discretion of the components' contributor. For instance, despite its Linux kernel and source code availability, Android has some proprietary components, while iOS is clearly proprietary with an open kernel (Darwin).

In the technological perspective, licenses and open sources refer to code reuse and, therefore, they are one part of the organizational level of technological interfaces. Meanwhile, for complex software architecture like OS, the principle of information hiding (Parnas, 1972; Baldwin and Clark, 2000) – via interfaces – must be put into perspective. In fact, documenting interfaces between third-party digital modules is a source of knowledge disclosure, particularly thanks to the SDK[10]. For instance, the cheap access to well documented SDKs could explain the fast growth of most of the online mobile application stores. This aims to match open source software projects, where "*the 'no hiding' principle allows developers to undertake much more sophisticated software engineering activities, such as redefining modules and interfaces specifications in response to the emergence of new interdependencies between separate modules. This is often the case in the introduction of radically new or substantially complex features in stable projects*" (Narduzzo and Rossi,

[9] "*Copyleft is a general method for making a program (or other work) free, and requiring all modified and extended versions of the program to be free as well*" (Source: http://www.gnu.org/copyleft/copyleft.html).

[10] The Software Development Kit gives the main tools and information for producing applications, services and contents complementary with the platforms.

2003, p. 27). However, not all knowledge is transferable, *qua* information (Nelson and Winter, 1982, chapter 4); it requires a certain amount of translation via a shared codebook, as well as the skills needed to understand it. Grimaldi and Torrisi (2001, p. 1427) apply this idea to the software industry: knowledge is "*articulated codified*" and "*unarticulated codified*" when, the codebook is, respectively, transparent among epistemic communities or inside one epistemic community; and knowledge is "*unarticulated uncodified*" when located in individuals (leaders), organizational routines or processes. That is why, even with open sources, the level of contribution depends on the experience in terms of previous contributions and past specialisation of the contributors (von Krogh *et al.*, 2003). Meanwhile, in the case of industries devoted to multitechnology and multicomponents products, Brusoni *et al.* (2001, p. 600) encourage us to discuss the boundaries of epistemic communities, because of the "*gap between what they make and what they know*". This gap could explain why Microsoft managed to make profitable collaborations with Mobile Network Operators (MNO), which have the necessary skills and exercise power all over the mobile industry (Evans *et al.* 2006, p. 195), in order to improve its mobile OS.

Finally, hierarchic organizations are also designed to limit fragmentation. For instance, Tee (2010) reminds us that during the Symbian Ltd period, cooperation on the Symbian OS and competition on the user interface (UI) induced a fragmentation of the system for end-users. Consequently, Symbian OS and UI were re-integrated, when Nokia acquired the full ownership of Symbian, then directed by an "architecture council" after Nokia transferred the ownership to the Symbian Foundation. This council is an interesting example of authority inside open source community, since "*the Foundation will operate as a meritocracy, with board and council membership allotted based on contribution to the platform*"[11]. In fact, O'Mahony and Ferraro (2007) show that hierarchies – based on meritocracy – emerge and evolve inside the open source communities. Meanwhile, for these communities, Raymond (1999) puts forward two organizational structures: hierarchical, *the Cathedral*, which involves few or more stakeholders according to the stage of software development; and, non-hierarchical, *the Bazaar*, when participation is open to everybody, all the time. Equally, the Google-Android example reminds us that, in the presence of sponsors, the interaction quality changes (Shah, 2006; West and O'Mahony, 2008). More precisely, the platform sponsors face a "*tension between control and openness*" (West and O'Mahony, 2008, p. 155): thanks to *control*, the platform sponsors "*assure ongoing alignment between their investment in the community and related product goals*", while with *openness*, they "*win greater external participation and technological adoption*". Equally, they distinguish two types of openness (*ibid.*, p. 151), in the case of sponsor-based organizational architectures: *transparency*, which "*allow[s] to understand what is happening*" and "*allow[s] use of the final product, the source code*" and; *accessibility*, which "*allow[s] to influence the direction of the community*".

[11] Source: Foundation website, offline from now on.

3.3 The Mobile OS System as Competing Systems: The Standards Issue

With such a level of granularity, the organization providing one platform seems to be a very baffling nest of decision and interaction processes, performed on a case by case basis. This view is the opposite of the expectations of organizational platform: i.e., developing and giving the direction of a core platform, as well as giving the roadmap and support for the provision of complements. In concrete terms, the *system integrator* has to manage the standards-setting and selection process to provide the technical compatibility enabling interorganizational coordination (Steinmueller, 2003) and, therefore, integration instead of fragmentation. Consequently, standards-setting is complex task. For us, standards are the top-level rules of the interfaces, meaning that they give global rules to shape the interfaces. Meanwhile, interfaces shape the standards, via organizational trade-offs. There is a co-evolution between technology and standards-setting (Funk, 2009). By this way, interfaces- and standards-setting is the cement that we use to link the technological and organizational level of abstraction.

In a nutshell, in Sections 2.2 and 2.3, we have seen how the technological side of interfaces results from technological interdependences. Then, in Section 3.2, we have seen how the technological side of interfaces has three types of organizational counterparts[12]: the definition of property rights, the active diffusion of technological knowledge and the hierarchical validation. Consequently, in this section we show how the economic and strategic concerns of the organization stakeholders shape the coevolution and, therefore, the standardization processes. Put concretely, the mobile service (as seen by end-users) "has to match with a quintuple layer of specifications" (Feijòo *et al.* 2009, p. 287): (i) the handset; (ii) the mobile OS; (iii) the application which runs the service; (iv) the wireless technology; and, (v) the operator's mobile system (portals, billing system …). That is why we split the standardization processes into two categories, whether they refer to the compatibility standards among layers inside one single platform; or, to the emergence of standards inside one layer.

In the first category, we focus on vertical standardization – in reference to the vertical integration of the system layer. In this case, Steinmueller (2003) uses the term "local standards", i.e. those standards "*that are unpublicised and used internally as a means of co-ordinating and dividing labour among different organisations*" (ibid., p. 135). In this line, this is easier to integrate the development processes of loosely coupled components, resulting from modular product architecture (Sanchez and Mahoney, 1996). Equally, standardized interfaces within the platform help integration without hierarchical authority, as integration produces increasing transaction costs when the stakeholder community and / or the module

[12] In some extent, these three dimensions come from the definition, by Pénin (2008), of an *open innovation context*.

quantity grow. This echoes the idea that standards make the validation process easier, by specifying rules form the beginning. These explicit rules – known ex ante by the stakeholders – reduce the hazard to having a contribution rejected by an authoritarian decision, once the work is already done or advanced. In fact, by specifying the interfaces, the standards give implicitly a list of compatible technological components. Furthermore, dynamically, standards give a relative inertia to the platform technologies. They shape the possible path of evolution of the platform. Put concretely, fragmentation may appear at one single mobile OS level, because of its evolutions via cyclical releases. The evolution of core and custom assets of a platform produces a compatibility issue at the platform level: backward and forward compatibility, i.e. according to the fact that a core platform release is compatible with functionalities of (respectively) older and newer applications. This point is ambivalent and encourages us to put forward some other critics of the vertical standardization strategy. Hence, in a platform related organization, made of potential competitors, this strategy is at the expense of differentiation. Furthermore, in firm networks, for Garud and Kumaraswamy (1995), the distinction between vertical complementors and competitors on the same layer become blurred because of knowledge sharing. The standardization process depends also on stakeholder trade-offs: either they encourage end-users to adopt the collective platform or else their own components.

In the second category, we focus on horizontal standardization, via inter-operability standards shared between platforms for specific layers. These standards are largely shared in consumer electronics and network industries, because of network externalities. In fact, the architectural innovation increases the interaction of organizations from the various layers and, therefore, modifies the competition structure. On each layer, we observe shared or competing horizontal standards. For instance, the mobile wireless layer has horizontal standards to allow the interoperability of these networks notwithstanding the specific mobile network operator chosen by a customer. Put generally, for some layer-specialized competitors, horizontal standardization on the other layers is a mean to make their entry on these layers easier; this is a sort of commoditization. For Steinbock (2003), since Nokia had no strong horizontal advantage on mobile OS, it encouraged collaboration on standards, in order to weaken competitors who staked everything on horizontal technology. Meanwhile, to weaken competitors, firms try to encourage their own standards' adoption and, therefore, reinforce their power. This strategy refers to standards wars, particularly when a firm uses its installed-base of customer to impose its standards (*de facto*), in order to prevent the entry of new competitors. This is different from *de jure* standardization when political governments and institution are the standards setters, like for the mobile wireless networks' standards. Beyond de facto and de jure, a third form of standardization is interesting in our case: this refers to the idea that the organizational side of the platform may act as a *standard committee* (Farell and Saloner, 1988). For instance, Nokia sponsored the Open Mobile Architecture Initiative (OAI), which gave birth to the OMA in 2002, by merging with the WAP Forum. This organization is in charge of the definition of mobile standards at every layer: it is a standard committee.

Equally, since March 2010, the WAC and LiMo have united their efforts, merging various mobile platform projects within an open architecture. This helps us understand the platform involvement of mobile network operators (MNO). In fact, via the WAC which an association of the main international MNOs and the technical support of LiMo (an open source Linux-based mobile OS), they aim to produce an open platform as open standard in order to remove fragmentation, which is detrimental for their revenue.

In our view, by rallying multitechnological vertical knowledge and adopting horizontal standards, the organizational side of the platform can produce collectively adopted standards and, thereby, provide system integration consistent horizontally, vertically and evolutionarily. For Meyer and Seliger (1998, p. 62), it “lies an even more important market advantage enabled by platform thinking and execution. If the developer builds and clearly communicates methods or techniques by which other companies or individuals can build modules that operate in or on the underlying platform, it has created the opportunity to become the standard or basis of large-scale innovation”. This strategy may rely on the SDK, particularly if this one is shared and produce inter-operability between platforms. However, in a previous paper, we had performed a case study of mobile OS, focusing on those provided by open source consortia (i.e. Android, Symbian by the Foundation and Linux Mobile, aka LiMo). The results had shown that the consortia leaders overtly want platform integrity by means of vertical cooperation. However: (i) the only tool to prevent fragmentation is the hierarchical selection for contribution; (ii) the legal structure authorizes free riding; (iii) sectoral concerns exhibit a preference for openness in term of “free beer” rather than “free speech”. Consequently, this does not guarantee the emergence of collective standards within the platform. Equally, about the horizontal standardization, strategic concerns at the firm level may outweigh collective concern at the platform or market levels. Hence, for a platform produced collectively by several firms, the standardization (or not) strategy relies mainly on the stakeholder opinion about the revenue source. When we have studied the technological side, it was relatively consistent to use the fitness concept as a real-valued dimension, in order to sketch the ability that a set of components has to fit into a technological environment, given their interdependences. However, when a firm estimates the value of its component as well as the value of the platform itself, economic concerns may be highly diverging among stakeholders. Even the definition of “value” differs: it does not necessarily refer directly to incomes, but to indirect incomes as an installed base of users, the customers' corporate image and so on.

4 Discussion: Some Implications for Computational Experiments

At this stage of our analysis, we have to articulate the set of proposed concepts and hypotheses within a formal system, in order to implement this system in simulation and perform computational experiments. Besides the growing use of computation in economics and, more generally, in social sciences, we draw some parallel with the use of computation in

computer science (via the paradigm of Evolutionary Computation, EC) and in engineering (via the evolutionary design). For Kicinger *et al.* (1995, pp. 4-5), "*evolutionary design is a branch of EC that integrates ideas from computer science (evolutionary algorithms), engineering (design science) and evolutionary biology (natural selection) to solve engineering design problems*". For them, "*the three main issues in applying EAs* [Evolutionary Algorithms] *to an engineering design problem are: [1] selecting an appropriate representation for engineering designs; [2] defining efficient genetic operators; [3] providing an adequate evaluation function for estimating the "fitness" of generated solutions (points in the search space)*". We refer to this sequence to organize the ideas.

4.1 Towards an Appropriate Representation

In line with Le Masson *et al.* (2009, p. 290), we consider a general framework where "*a collaborative process of platform design can actually be itself a specific platform; we shall call it a 'platform for platform design'*". Hence, there is a real interest in analysing these processes at the systemic level, from the very earliest stage, all the more so as the literature has only just started to address this question (Le Masson *et al.*, 2009; Maula *et al.*, 2006; West and Wood, 2008). In other words, for Henderson and Clark (1990), *architectural innovation* requires reorganization and the acquisition of knowledge by firms. That is why we use the "platform for platform design" concept as an abstraction to represent the organisation as an evolving entity. In some extent, it echoes the representation used by Brusoni and Prencipe (2009) where the product, organization and knowledge are represented via different domains / planes, which are not matching perfectly.

Hence, in our approach, unlike the NK models, we consider that two planes are useful to represent separately the product and organization dynamics. Then, we have show before, how the interface may be used to connect these two planes, actually with a third plane. The planes have the same dimensions, i.e. technological characteristics and service characteristics. In order to cope with the definitions of core and peripheral / complement / custom components, we cannot accept the traditional representation of modularity by a one-to-one mapping between service and technology. In fact, each service is connected with at least one core and one peripheral technological characteristic, i.e. technological characteristics mapping with (respectively) several or one service characteristic. Then, the interface plane connect the modules (i.e. non-overlapping sets of technological and service characteristics); Finally, the organization plane is cut to match the technological and service characteristics, as they are divided among the stakeholder of the organization in terms of division of labour on the platform.

4.2 The Genetic Operators of an Architectural Innovation

In this case, the genetic operators are shaped by the architectural innovation. Hence, unlike the operators analysed by Baldwin and Clark (2000) which refers to the modularization of components, we are interested be operators which modify the architecture with relatively

stable components. Consequently, genetic operators are related with the organisational side of interface- and standards-setting. That is to say that in our representation, the change is inseparable from the knowledge acquisition and diffusion; this has various levels, from the interfaces between components to the product shaping standards. This is the central part of the model and a place for several implementations of the computational experiments.

This part of the model lies on the idea that, facing an architectural innovation, the organization is mainly focused on architectural learning (Henderson and Clark, 1990), since the services offered by components are already know (at least by the component owner). Put concretely, the organization begins to work with the three given planes. The initial architecture does not match perfectly the “perfect” architecture, i.e. the interface architecture which offers the best results at the technological level. According to its own architecture (in terms of interface’s organizational counterparts, division of labour among firms), the organization try to improve the product architecture by testing new interfaces. Put concretely, one explore the product architecture, in order to discover potential interfaces which may improved the service characteristics of the platform (this interfaces are defined ex ante by the simulation model, but the organization as a whole does not know where they are).

In this line, several implementations and calibrations may be performed: e.g. the ability to experiment in parallel or sequentially; various levels of labour division (one single integrated firm, sponsor-based organization, community of firms …); a focus on reducing the amount of technology or on the service value; various preference for knowledge disclosure; and so on. For instance, an interesting issue is to study the link between the location of the interface into the product architecture, comparatively with the structure of the organization at this level, i.e. this refer to test the idea that we observe better results when highest-level interdependences are managed by highest-level integrators. Consequently, we imagine an operator related to move the position of a technology in the hierarchy by changing the interfaces: for core components what is the effect to, respectively, put the component up in the hierarchy; improve its connection within a single subsystem; give up this component?

These implementations refer to the first step of computational experiments, when we focus on one single platform. Consequently, the standardization process is only vertical via the attempt to fix some interfaces technologically and organizationally. The next step is of course the study of a market made up of several competing platforms which have eventually to choose for shared horizontal standards, not to say to promote their standards to the competitors. Another step is to analyse the incremental innovations at the component level, resulting from the new opportunities given by the architectural innovation. However, these implementations are not supported theoretically in this paper and, therefore, call for further formalizations.

4.3 Merging Various Fitness Representations

The modelling strategy based on fitness value as the main index of decision has been

recently criticized, because – unlike biological organisms – economic organizations have other concerns. That is to say, we cannot use the same index to compute a success on the organizational side and on the technological side. By this way, some recent simulation models (e.g. Ciarli et al., 2008; Marengo and Valente, 2010) focus on model of markets, where the supplying organizations build their decisions regarding the effect on the demand side. Clearly, this option is interesting for our project eventually. However, we assume that some experiments have first to be implemented on the supply side, in order to understand better the whole behaviour of this architectural exploration model.

Consequently, the aim of this model is not to produce policy implications according to the "best" organizational architecture to cope with architectural innovation. Conversely, we focus on the "costs of experimentations" involved by various organizational architecture exploring various technological architecture. These costs are diverse: e.g. the level of necessary knowledge disclosure, the time spent to reach a good level of architectural innovation, the probability to be locked into sub-optimal solutions and so on. For instance, a product platform managed by a single proprietary firm is prone to reduce its technological opportunities because of the lack of external or specialized knowledge. Meanwhile, this structure may benefit from authority, while community based platform may suffer from fragmentation, divergent decisions and so on.

Put concretely, the stakeholders within the organization compute some potential fitness, according to their knowledge of the platform architecture. They focus either on improving the platform fitness as much as possible, or on improving their piece of the platform only. By this way, we observe the "costs" related to these improvements via the indexes proposed above.

5 Conclusion

Consequently, thanks to this model, we aim to test varying strategies in terms of standardization and knowledge accessibility (related with various levels of organizational integration) according to the involvement of firms to reduce the technological complexity of the system (i.e., increasing the modularity of the OS). Hence, one contribution of this paper is the articulation of technologies and functionalities in an applied perspective of complexity, in order to correspond better with the architectural view followed by mobile OS projects' stakeholders. The main contribution is about the collective construction of the architecture linking these technologies and functionalities. In concrete words, we aim to show the opportunities and limits of integrated and de-integrated organizations, according to their involvement in sharing knowledge (and, consequently, technologies) vertically (i.e. sharing functionalities) and horizontally (i.e. adopting standards and / or sharing interfaces for the system core).

References:

ALTENBERG, L. (1994), "Evolving better representations through selective genome growth", *The 1st IEEE Conference on Evolutionary Computation*, pp. 182-187

ALTENBERG, L. (2005), "Modularity in evolution: some low-level questions", in Callebaut, W., Rasskin-Gutman, D. (Eds), *Modularity: Understanding the Development and Evolution of Natural Complex Systems*, The MIT Press, pp. 99-128

ANTONELLI, C. (2005), "Models of knowledge and systems of governance", *Journal of Institutional Economics*, vol. 1 (1), pp. 51-73

BALDWIN, C. Y., CLARK, K. (2000), *Design Rules - The Power of Modularity*, MIT Press, Cambridge

BALDWIN, C. Y., CLARK, K. (2006), "The architecture of participation: does code architecture mitigate free riding in the open source development model?", *Management Science*, vol. 52 (7), pp. 1116–1127

BALDWIN, C. Y., WOODARD, C. J. (2009), "The architecture of platforms: a unified view", in GAWER, A. (Ed.), *Platforms, Markets and Innovation*, Edward Elgar, Cheltenham, pp. 19-44

BLUME, M., APPEL, A. W. (1999), "Hierarchical modularity", *ACM Transactions on Programming Languages and Systems*, vol. 21 (4), pp. 813-847

BONACCORSI, A., GIANNANGELI, S., ROSSI, C. (2006), "Entry strategies under competing standards: Hybrid business models in the open source software industry", *Management Science*, vol. 52 (7), pp. 1085-1098

BRUSONI, S., PRENCIPE, A., PAVITT, K. (2001), « Knowledge specialization, organizational coupling, and the boundaries of the firm: why do firms know more than they make?", *Administrative Science Quarterly*, vol. 46 (4), pp. 597-621

BRUSONI, S., MARENGO, L., PRENCIPE, A., VALENTE, M. (2004), "The value and costs of modularity: a cognitive perspective", *SPRU Electronic Working Paper Series*, n°123

BRUSONI, S., PRENCIPE, A. (2009), "Design rules for platform leaders", in GAWER, A. (Ed.), *Platforms, Markets and Innovation*, Edward Elgar, Cheltenham, pp. 306-321

BRUSONI, S., FONTANA, R. (2011), "Incumbents' strategies for platform competition – Shaping the boundaries of creative destruction", in DE LISO, N., LEONCINI, R. (Eds), *Internationalization, Technological Change and the Theory of the Firm*, Routledge, pp. 66-88

BUENSTORF, G. (2005), "Sequential production, modularity and technological change", *Structural Change and Economic Dynamics*, vol. 16, pp. 221-241

CIARLI, T., LEONCINI, R., MONTRESOR, S., VALENTE, M. (2008), "Technological change and the vertical organization of industries", *Journal of Evolutionary Economics*, vol. 18, pp. 367-387

DANILOVIC, M., BROWNING, T.R. (2007), "Managing complex product development projects with design structure matrices and domain mapping matrices", *International Journal of Project Management*, vol. 25 (3), pp. 300-314

DOSI, G., LEVINTHAL, D. A., MARENGO, L. (2003), "Bridging contested terrain: linking incentive-based and learning perspectives on organizational evolution", *Industrial and Corporate Change*, vol. 12 (2), pp. 413-436

ENGLER, D.R., KAASHOEK, M.F., O'TOOLE, J. Jr (1995), "Exokernel: An operating system architecture for application-level resource management", *Proceedings of the fifteenth ACM symposium on Operating systems principles*, pp. 251-266

ETHIRAJ, S.K., LEVINTHAL, D. (2002), "Search for architecture in complex worlds: an evolutionary perspective on modularity and the emergence of dominant designs", *Working Paper*

ETHIRAJ, S.K., LEVINTHAL, D. (2004), "Modularity and innovation in complex systems", *Management Science*, vol. 50 (2), pp. 159-173

EVANS, D.S, HAGIU, A., SCHMALENSEE, R. (2006), *Invisible Engines – How Software Platforms Drive Innovation and Transform Industries*, The MIT Press, Cambridge

FARRELL, J., SALONER, G. (1988), "Coordination through committees and markets", *RAND Journal of Economics*, vol. 19 (2), pp. 235-252

FEIJÒO, C., MAGHIROS, I., ABADIE, F., GÒMEZ-BARROSO, J. L. (2009), "Exploring a heterogeneous and fragmented digital ecosystem: Mobile content", *Telematics and Informatics*, vol. 26 (3), pp. 282-292

FRENKEN, K. (2006), *Innovation, Evolution and Complexity Theory*, Edward Elgar, Cheltenham

FUNK, J. L. (2001), *The Mobile Internet: How Japan Dialed Up and the West Disconnected*, ISI publications, Hong Kong

FUNK, J.L. (2009), "The co-evolution of technology and methods of standard setting: the case of the mobile phone industry", *Journal of Evolutionary Economics*, vol. 19 (1), pp. 73-93

GANCO, M., HOETKER, G. (2009), "NK modeling methodology in the strategy literature: bounded search on a rugged landscape", in BERGH, D., KETCHEN, D. (eds.), Research *Methodology in Strategy and Management*, Emerald Group Publishing, Ltd.

GARUD, R., KUMARASWAMY, A. (1995), "Technological and organizational designs for realizing economies of substitution", *Strategic Management Journal*, vol. 16 (1), pp. 93-109

GAWER, A. (Edited by) (2009), *Platforms, Markets and Innovation*, Edward Elgar, Cheltenham

GRIMALDI, R., TORRISI, S. (2001), "Codified-tacit and general-specific knowledge in the division of labour among firms - A study of the software industry", *Research Policy*, vol. 30, pp. 1425–1442

HENDERSON, R.M., CLARK, K.B. (1990), "Architectural innovation: the reconfiguration of existing product technology and the failure of established firms", *Administrative Science Quarterly*, vol. 35, pp. 9-30

KAUFFMAN, S. A. (1989), "Adaptation on rugged fitness landscapes", *Lectures in the Sciences of Complexity*, vol. 1, pp. 527-618

KELLY, J. (2006), "Design strategies for future wireless content", in GROEBEL, J., NOAM, E. M., FELDMANN, V., (Eds.), *Mobile Media - Content and Services for Wireless Communications*, Lawrence Erlbaum, Mahwah, pp. 69-85

KICINGER, R., ARCISZEWSKI, T., JONG, K.D. (2005), "Evolutionary computation and structural design: A survey of the state-of-the-art", *Computers & Structures*, vol. 83 (23-24), pp. 1943-1978

LANCASTER, K.J. (1966), "A new approach to consumer theory", *Journal of Political Economy*, vol. 14, pp. 133-156

LANGLOIS, R.N (1999), "Modularity in technology, organization, and society", *Department of Economics Working Paper Series*, vol. 5

LE MASSON, P., WEIL, B., HATCHUEL, A. (2009), "Platforms for the design of platforms: collaborating in the unknown", in GAWER, A. (Ed.), *Platforms, Markets and Innovation*, Edward Elgar, Cheltenham, pp. 273-305

LERNER, J., TIROLE, J. (2005), "The scope of open source licensing", *The Journal of Law, Economics, & Organization*, vol. 21 (1)

LUER, C., ROSENBLUM, D. S., VAN DER HOEK, A. (2001) "The evolution of software evolvability", *Proceedings of International Workshop on the Principles of Software Evolution*, Vienna, Austria, pp. 131–134

MACCORMACK, A., VERGANTI, R. (2003), "Managing the sources of uncertainty: Matching process and context in software development", *Journal of Product Innovation Management*, vol. 20 (3), pp. 217-232

MACCORMACK, A., RUSNAK, J., BALDWIN, C. Y. (2006), "Exploring the structure of complex software designs: an empirical study of open source and proprietary code", *Management Science*, vol. 52 (7), pp. 1015–1030

MACCORMACK, A., RUSNAK, J., BALDWIN, C.Y. (2008), "Exploring the duality between product and organizational architectures: A test of the mirroring hypothesis", *Working Paper*, Harvard Business School, 08-039

MAULA, M., KEIL, T., SALMENKAITA, J.-P. (2006), "Open innovation in systemic innovation contexts", in CHESBROUGH, H., VANHAVERBEKE, W., WEST, J. (Eds), *Open Innovation: Researching a New Paradigm*, Oxford University Press, Oxford, pp. 241-257

MARENGO, L., VALENTE, M. (2010), "Industry dynamics in complex product spaces: An evolutionary model", *Structural Change and Economic Dynamics*, vol. 21 (1), pp. 5-16

MARENGO, L., DOSI, G. (2005), "Division of labor, organizational coordination and market mechanisms in collective problem-solving", *Journal of Economic Behavior & Organization*, vol. 58 (2), pp. 303-326

MEYER, M. H., SELIGER, R. (1998), "Product platforms in software development", *Sloan Management*

Review, vol. 40 (1), pp. 61-74

MURMANN, J.P., FRENKEN, K (2006), "Toward a systematic framework for research on dominant designs, technological innovations, and industrial change", *Research Policy*, vol. 35 (7), pp. 925-952

NALEBUFF, B. J., BRANDENBURGER, A. M. (1997), *Co-opetition*, Harper Collins Business, London

NARDUZZO, A., ROSSI, A. (2003), "Modular design and the development of complex artefact lesson from free open source software", *Quaderni DISA*

NELSON, R. R., WINTER, S. (1982), *An Evolutionary Theory of Economic Change*, The Belknap Press of Harvard University Press, Cambridge

O'MAHONY, S., FERRARO, F. (2007), "The emergence of governance in an open source community", *Academy of Management Journal*, vol. 50 (5), pp. 1079–1106

PARNAS, D.L. (1972), "On the criteria to be used in decomposing systems into modules", *Communications of the ACM*, vol. 15 (12), pp. 1053-1058

PÉNIN, J. (2008), "More open than open innovation? Rethinking the concept of openness in innovation studies", *Documents de Travail*, n° 18, BETA (working paper)

RAYMOND, E. (2001), *The Cathedral and The Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly Media, Sebastopol (CA)

RIVKIN, J.W., SIGGELKOW, N. (2003), "Balancing Search and Stability: Interdependencies among Elements Organizational Design", *Management Science*, vol. 49 (3), pp. 290-311

RIVKIN, J.W., SIGGELKOW, N. (2007), "Patterned interactions in complex systems: Implications for exploration", *Management Science*, vol. 53 (7), pp. 1068-1085

SANCHEZ, R. (1995), "Strategic flexibility in product competition", *Strategic Management Journal*, vol. 16 (1), pp. 135-159

SANCHEZ, R., MAHONEY, J.T. (1996), "Modularity, Flexibility, and Knowledge Management in Product and Organization Design", *Strategic Management Journal*, vol. 17, pp. 63-76

SAVIOTTI, P.P., METCALFE, J.S. (1984), "A theoretical approach to the construction of technological output indicators", Research Policy, vol. 13, pp. 141-151

SEN, R., SUBRAMANIAM, C., NELSON, M. L. (2008), "Determinants of the choice of open source software license", *Journal of Management Information Systems*, vol. 25 (3), pp. 207–239

SHAH, S. K. (2006), "Motivation, governance, and the viability of hybrid forms in open source software development", *Management Science*, vol. 52 (7), pp. 1000-1014

SIGGELKOW, N., LEVINTHAL, D. (2003), "Temporarily divide to conquer: centralized, decentralized, and reintregated organizational approaches to exploration and adaptation", *Organization Science*, vol. 14, pp. 650-669

SIMON, H. A. (1962), "The architecture of complexity", *Proceedings of the American Philosophical Society*, vol. 106 (6), pp. 467-482

STEINBOCK, D. (2003), *Wireless Horizon: Strategy and Competition in the Worldwide Mobile Marketplace*, Amacom Books, New York

STEINMULLER, W. E. (2003), "The role of technical standards in coordinating the division of labour in complex system industries", in PRENCIPE, A., DAVIES, A., HOBDAY, M., *The Business Of Systems Integration*, Oxford University Press, Oxford, pp. 133-152

TEE, R. (2010), "Coordinating technological collaboration in fast changing environments: understanding the interplay between product and organizational architecture", *DRUID Summer Conference 2010*

VALENTE, M. (2008), "Pseudo-NK: an enhanced model of complexity", *LEM Papers Series*, vol. 26

VON HIPPEL, E., VON KROGH, G. (2003), "Open source software and the 'private-collective' innovation model: issues for organization science", *Organization Science*, vol. 14 (2), pp. 208-223

VON KROGH, G., SPAETH, S., LAKHANI, K.R. (2003), "Community, joining, and specialization in open source software innovation: a case study", *Research Policy*, vol. 32 (7), pp. 1217-1241

VON KROGH, G., STUERMER, M., GEIPEL, M., SPAETH, S., HAEFLIGER, S. (2009), "How component dependencies predict change in complex technologies", *Druid Summer Conference 2009*

WEST, J. (2003), "How open is open enough? Melding proprietary and open source platform strategies", *Research Policy*, vol. 32, pp. 1259–1285

WEST, J., O'MAHONY, S. (2008), "The Role of participation architecture in growing sponsored open source communities", *Industry and Innovation*, vol. 15 (2), pp. 145-168

WINTER, S.G., CATTANI, G., DORSCH, A. (2007), "The value of moderate obsession: Insights from a new model of organizational search", *Organization Science*, vol. 18 (3), pp. 403-419

WOODARD, C. J. (2008), "Platform competition in digital systems: architectural control and value migration", *Working Paper*

YAKOB, R., TELL, F. (2007), "Managing near decomposability in complex platform development projects", *International Journal of Technology Intelligence and Planning*, vol. 3 (4), pp. 387-407

YU, L., RAMASWAMY, S. (2006), "Software and biological evolvability: a comparison using key properties", *Second International IEEE Workshop on Software Evolvability*, pp. 82-88