Alain Sandoz (University of Neuchâtel, Switzerland) & Léa Stiefel (University of Lausanne, Switzerland

Untying the knot between software-based platforms and information infrastructures

Keywords: inter-organizational platform, information infrastructure, shared functionality, service platform, intermediation platform, interoperability platform

1. Introduction

Software-based platforms (Tiwana *et al*., 2010) are important parts of information infrastructures (Hanseth and Bygstad, 2021). There is however a knot that remains to be untied to fully understand how platforms might *drive* their emergence, structuring, and architecting. It appears from the literature that *all software-based platforms are centrally controlled by an owner*, and, therefore, that information infrastructures (*II*s) might be made up, at best, of constructs or juxtapositions of centralized platform ecosystems.

In this paper, we intend to untie this knot and free up some space for research on alternatives to a world of *II*s that would be an archipelago of isolated island-states and of federations of platforms under the control of public or private entities jockeying for power.

The concept of platform-oriented infrastructure (Hanseth and Bystad, *ibid*) describes what is to our knowledge the most elaborate way of *articulating* large digital platforms in order to reach higher infrastructural levels. Drawing on the seminal work of Tiwana (2014) and on contributions from many authors, the authors write, in relation to current research on platforms: "*Platform ecosystems and infrastructures are similar in the sense that they include a huge number of technological components as well as developers and development organizations, ... [but] there are also significant differences: platform ecosystems are all based on one specific architecture in terms of the split between platform and apps, and a specific governance structure where* ***one single actor owns and controls*** *the platform while autonomous app developers control the apps*". In his book review of (Tiwana, *ibid*), Kumar (2018) summarizes: "*The* ***platform owner must*** *achieve autonomy of app developers and also integration of efforts of individual contributors. These twin goals can be achieved by an appropriate mix of decision rights,* ***control mechanisms****, and pricing policies*".

We do not share this view. Software-based platforms may be neither proprietary nor centrally controlled, and some fundamental platforms of the Internet are. In this paper, we identify such platforms, study their properties, and discuss why they are interesting. We will consider only inter-organizational interactions and how information systems (*IS*s) of organizations interoperate on platforms.

Throughout the paper, we consistently use the terms interact/interaction between organisations (*i.e*., module- and *IS*-owners) and interoperate/interoperation between information systems and/or modules and/or programs. We purposely leave aside end-users (*e.g*., the private persons that use apps running on their smart-phone) and the corresponding platform ecosystems (*e.g*., Android OS or Apple iOS). As our examples show, this does not restrict the generality of our position: we do not pretend that centrally controlled platforms should not be parts of *II*s, only that they should not be the *only* parts.

The paper is structured as follows. We first dissect the definition of software-based platform from (Tiwana *et al*., *ibid*) and isolate its *core shared functionality*. We show that TCP/IP is a software-based platform in this sense, which is neither proprietary, nor centrally controlled. In the following sections, we identify different forms of core shared functionality and examine each form separately: service platforms (*by definition* are centralized and proprietary), intermediation platforms (*by default*, are decentralized), interoperability platforms (*by construction* are *fully* distributed). We then study the relationships *between* platforms. We make a distinction between platform- and information system- dependencies and show that dependencies between platforms might be relayed in different ways to the *IS*s that operate on these platforms. Finally, we come back to the analogical world and to our initial objective of *untying the knot*. We conclude with a discussion of questions raised by our perspective.

2. The core shared functionality of a platform

Tiwana *et al*. (*ibid*) defines a *software-based* platform as "the *extensible codebase* of a software-based system that provides core functionality shared by the modules that interoperate with it and the interfaces through which they interoperate". This definition is central to their research note on platform-centric ecosystems and is used consistently in much of the research that has followed over the years on the relationship between platforms and *II*s. We make three remarks on this definition: 1) the definition does not specify whether the *software-based system* is centrally controlled or not. The system could be centralized, decentralized, or distributed; 2) it is the *core functionality* that is shared, not the *extensible codebase*; and 3) the interfaces are an integral part of the platform, and the modules use them to interoperate with the *software-based system* (*i.e*., not specifically with the *IS* of some actor).

In all generality, modules are programs, and interfaces are application programming interfaces (APIs). If 1) some software-based system (*e.g*., the enterprise *IS owned* by some actor) were 2) to provide a *core functionality* (implicitly shared) to external modules, 3) together with the APIs through which the modules would interoperate with that *IS* (or more precisely with the services of the *IS* that provide the functionality by executing the *codebase*), then that "platform" would *by definition* be proprietary and centrally controlled.

But, let's imagine that a distributed software-based system was to provide some core shared functionality to a collection of modules, and that the owners of those modules each controlled the API through which their module would interoperate with that distributed system. Provided (for consistency) that the codebase was extensible, then that system would be a software-based platform in the sense of (Tiwana et al., ibid).

If the component of the system that executes the *portion of the codebase* used to implement the functionality for a module, *and* its API, were both *locally* controlled by the module's owner; and if there was no central component in the system, ***then*** the platform might be *fully distributed*, and neither proprietary nor centrally controlled.

The codebase that implements the TCP/IP stack and that is *executed* throughout the Internet by programs running on computer systems from within information systems (*i.e*., *modules*) is a software-based platform. Although there are many implementations of TCP/IP, at any time any computer system that is connected to the Internet (*i.e*., any *IP-host*) must locally operate at least one and control its API. The codebase is *extensible*. New implementations of TCP/IP can be easily produced (if only in higher education engineering institutions) and extensions of the underlying protocols, *e.g*., from IPv4 to IPv6, happen from time to time, although this kind of transition proves laborious on a socio-technical level (as shown by DeNardis, 2009). The set of all IP-hosts is a (distributed) software-based system that provides a shared core functionality through this codebase. The functionality is *end-to-end controlled transmission of data between programs* that run on computer systems. It is shared by these modules. The API through which a program interoperates with this system is the TCP/IP system-call interface of the computer on which it is executed.

The TCP/IP software-based platform is neither centrally controlled, nor proprietary. A skilled programmer might modify the portion of the codebase under his/her control, *i.e*., *hack* the TCP/IP implementation of her/his Linux OS, recompile the kernel, and run programs on the computer system as before. Provided this version respects the requirements of TCP/IP when it interoperates with other IP-hosts, probably no one will notice. Note that the *codebase is not shared*. What is shared is the *core functionality* of TCP/IP, *i.e*., the generalized capability to *inter-operate* conferred to any pair of programs running at any given time on IP-hosts. TCP/IP is a *low-level* platform that resides in the *transport layer* of the Internet, so low that it remains under the radars. We will see higher level examples, *i.e*. platforms in the *application* layer, later.

3. Service platforms

We call the type of platform usually understood in the sense of (Tiwana *et al*., *ibid*) a *service platform*. The core functionality *shared* by modules is a bundle of services provided by a servicing information system (*SIS*) through its APIs, *i.e*., interfaces that the *SIS* controls. Modules *external* to the *SIS* use the services but do not directly *interoperate*: data only flows between the *SIS* and each module's *IS*. Nevertheless, if its service provides a functionality like transaction consistency (*e.g*., as for SWIFT - Scott and Zachariadis, 2012), the *SIS* might *relay* data between the client-*IS*s. So, depending on the service, client organizations might not interact at all, or interact only indirectly over the platform.

Although an *IS* might be a large and complex system, it is usually considered to be operated under the control of a single entity that owns it. The owner's *management* sees control of the *IS* as a strategic priority and drives it according to principles of *enterprise architecture* (Hanseth and Bygstad, *ibid*; Ross *et al*., 2006). The platform ecosystem built on top of such a client-server configuration will be *centrally controlled* by the service provider; its software base and APIs will be proprietary.

Service platforms might be combined in several ways. The *SIS* of one platform might use the shared functionality of another *SIS* through the latter's APIs, and *vis versa*. This need not be restricted to a pair of service platforms: any number of *SIS*s can be connected by a network of direct client-server relations. The network might be centralized, or decentralized, or locally distributed. To articulate service platforms in this way enables to develop platform- oriented *II*s in the sense of (Hanseth and Bygstad, *ibid*). Multiple bilateral interconnections however stumble on the problem of the standardization of data. This is a research topic of its own (Hanseth *et al*. 1996; Poppe *et al*., 2014; Sæbø and Poppe, 2015; Nielsen and Sæbø 2016).

Nielsen and Aanestad (2005) report on another type of combination: two platform owners who dominated the mobile networks in Norway collaborated to provide services to the same third-party developers through the APIs of their respective *SIS*s. In this way they drove the emergence of a unique market for mobile content- providers and -consumers, that they together controlled.

For module owners, the client-server configuration can be problematic, as the literature has shown for platforms like Twitter (Bucher, 2013; Puschmann, 2013) or Facebook (Helmond, 2015). In addition to functional, causal and technical dependencies of a client-*IS* on the *SIS*, to control the API gives the *SIS owner* indirect control on what a module *owner* can do (control of activity), how (control of semantics), and when (control of temporality) (Stiefel and Sandoz, 2022). These master-slave dependencies apply at the level of organisations. If they remain unbalanced, *i.e*., without clear incentives (business) or guarantees (through governance), then the platform may be rejected (Stiefel, *in press*).

4. Intermediation platforms

Another type of software-based *PF* is an *intermediation platform*. These platforms provide *support functionalities* to modules. They have dedicated interfaces, that might or might not be controlled by module owners.

The *implementation* of the Internet *internet layer*, the Internet Domain Name System, and *blockchains* are intermediation *PF*s.

The Internet *internet layer* is composed of information systems that implement the transport of TCP/IP packets through routers and over physical links. Each router is controlled by a unique Internet Service Provider (ISP). Links are controlled by multiple ISPs using contracts. *Transport* within this global physical network is a core functionality provided by *ISP*s (*i.e.*, enterprises, each under the jurisdiction of one state) that are directly connected point to point through local area networks (*LAN*s) or backbones using their own fully controlled interfaces. The banal end-user Internet-host does not share this core functionality with *ISP*s. It uses the transport function only on the last *link*, *i.e.*, on the *LAN* that connects it to its *ISP*. Transport in the open Internet is only a support function for communication between modules. The *ISP* enterprises, possibly under the pressure of states or of big customers, or on arbitrary grounds, can filter, slow down or block out, decipher and read, etc. traffic that transits through their routers (DeNardis, 2012).

*ISP*s *collectively* operate a *private information infrastructure* to manage *routing data* for the transport of TCP/IP packets according to their policies.

The Internet Domain Name System (*DNS*) (Mockapetris, 1983) provides name resolution in the Internet to programs that need to access remote resources. For example, any program that sends a request to a web server must have resolved the domain name of the URL before connecting to the server. The *DNS* is a software-based platform composed of a decentralized hierarchy of servers that resolve names in specific areas of the Internet, either based on local knowledge (exact or cached) or by forwarding unresolved requests down the hierarchy, and responses back up. The *DNS-PF* is decentralized and some of its nodes are controlled, in particular, by *ISP*s (using other policies than for transport). Control and ownership of the *DNS* are decentralized. The *DNS* support function can be altered under the pressure of states and attacked in many ways (Musiani, et al. 2016; in particular, Musiani, 2016; and Merrill, 2016).

Finally, blockchains (*BC*s) are software-based platforms that provide the support functionality of *ordered consensus on block contents*. Any module may submit requests to a *BC* for the execution of code (smart contracts) over APIs that it controls locally. But only blockchain- *nodes* enforce consensus. Blockchains are in general non- proprietary and control is decentralized. A blockchain might be more or less open (from fully open, *e.g.*, Bitcoin, to permissioned with unequal voting rights, *e.g.*, Hyperledger Fabric).

Modules that use an intermediation platform, do not interoperate on the platform and module-owners do not interact when they use it. The platform only *supports* some possible interaction. Modules might suffer 1) from failures of the support function; and in the case of blockchains, 2) from unexpected temporal dependencies due to the total ordering algorithm used by the *BC* nodes.

5. Interoperability platforms

The last type of software-based *PF* is an *interoperability platform*. On these platforms, modules use the shared function to *interoperate directly*, *i.e.*, interoperability platforms support direct interaction between module owners. The global implementation of TCP/IP, seen as a software-based system, is the most widespread interoperability platform: it underlies the Internet information infrastructure, and in fact any digital *II*. The platform is a distributed peer-to-peer (*P-2-P*) system. Every IP-host can communicate with any other over TCP/IP, provided both agree (and the underlying support transport function does not fail). Each IP-host owner is free of its associations and can connect its *IS* to the platform, or disconnect it, at any time. The *IS* fully controls the execution of the codebase it uses and the APIs that give access to this codebase.

TCP/IP lies in the intermediary *transport layer* of the Internet, above the *internet layer* and below the *application layer*. The latter contains many software- based platforms, including *interoperability* platforms such as FTP or HTTP. Even though these names designate protocols, the codebase that implements these protocols, the distributed system where this codebase is executed, together with the APIs through which modules access their local version of the codebase, compose software- based platforms whose core shared functionalities enable modules to interoperate directly.

In the examples above, the organizations that own and operate an *IS* using the platform are *peers*. Their roles and responsibilities with regards to the platform and its usage are assumed in all freedom and perfectly symmetrical. It is important to place these considerations at the organizational level, because when the core shared functionality is eventually used (*e.g.*, for a file transfer via the FTP platform), the technical relationship between modules might be client-server. The role of being slave or master, and the choice of the master, *resp*. slaves to answer to, is however a choice of the peer, and it can be played in the opposite direction anytime. The *P-2-P* concept generally supposes that peers share a common resource, *e.g.*, files, computing power, bandwidth, etc. (Méadel and Musiani, 2015; Musiani, *ibid*). On interoperability platforms, the core shared functionality *is* the common resource, not the data that transits (packets, files or contents) when peer-*IS*s interoperate.

Based on fieldwork conducted in 2018 and 2019, Stiefel and Sandoz (2021) study the case of an interoperability platform that was a *digital commons* (Stalder, 2010). The shared functionality was *P-2-P* data transmission between organisations that operated a database in Swiss agriculture, provided that it was authorized by the farmer who owned the data. The codebase of the platform was opensource. It implemented a set of services packaged into a *generic node*. Each peer (organization) operated and controlled its own node and the APIs through which it invoked the functionality. The function required specific mechanisms in order to guarantee asynchrony (each party remained temporally autonomous) and support autonomy, liberty of association, and trustworthiness of peers. The *capacity to exchange data* was the common resource of the platform, not the farmers' data. Its usage was defined by principles, rules, and requirements from the environment. Modules that used the platform depended on its mechanisms (*i.e.*, on its codebase) and module-owners on its rules of usage, but no interorganizational dependencies were induced by the platform (Stiefel and Sandoz, 2022).

A second case that we studied was individual traceability of animals in livestock. The traceability of an object is the capability to establish a *chain of events* that guarantees some property of a given object at some instant (*e.g.*, some animal has never received AB treatment). This is done by proving that the property is stable when any event of the chain occurs and between any pair of events; and by following the given chain *back up* to some point where the property *was* known to be true. Different actors might be interested in different properties of the same objects and use different events to establish the properties they are interested in. Different chains of events might not go through the same locations, and not reach common destinations in the same order. Because events first occur and are then reported (after occurrence), total ordering (*e.g.*, using a *BC*) scales up poorly. We believe that an interoperability platform designed to realize traceability by implementing *transmission of events and delivery at destination in causal order* (Schiper *et al.*, 1989), is feasible and would have the capacity to scale up well.

6. Dependencies between platforms, scaling up

To summarize, the examples we gave of different types of platforms show that:

- concerning dependencies imposed on modules/owners 1) service platforms impose dependencies on module- owners in favour of platform-owners; 2) an intermediation platform might induce dependencies of modules on the platform (*e.g*., through the failure of the support function), and possibly indirect dependencies between modules (*i.e*., temporal dependencies due to the total ordering of blocks by a *BC*). Inter-module dependencies might be relayed to module-owners; and 3) inter-operability platforms that are designed *P-2-P* with liberty of association between peers seem to not *by themselves* impose dependencies to module-*owners*.
- concerning dependencies of platforms 1) interoperability platforms (TCP/IP, HTTP) can depend on intermediation platforms (*e.g*., transport in the Internet, *resp*. *DNS*). Intermediation platforms like the two latter might be locally controlled, *e.g*., by *ISP*s, and suffer from political constraints; and 2) service or intermediation platforms can depend on interoperability platforms (*e.g*., Facebook and blockchains depend on TCP/IP).
- concerning the ability to technically scale up (scalability) 1) the scalability of a service platform depends on the interest and the capacity of the platform owner to sustain demand and the growth of its *SIS*; 2) the scalability of an intermediation platform depends on its organization and on the support function (*DNS* was built to scale, whereas blockchains in general have problems to scale); 3) the scalability of an interoperability platform seems to depend more on the complexity of the *meta*-data necessary to manage the shared functionality, rather than directly on the functionality itself (*e.g*., TCP/IP).

7. Back to the analogical world

In the analogical world, platforms, though not *software based*, have long emerged to support interaction by providing *shared* functionalities to actors. 1) Language (for *direct* oral or written communication), 2) currencies,
3) fax (for legal document exchange), 4) dictionaries (to *support* actors using different languages), 5) stock exchanges (for trade), and 6) deep sea harbours (for transport), are examples of analogical platforms (that we assimilate *resp*. to interoperability (1-3), intermediation (4), and service (5, 6) platforms).

Users evolve and platforms adapt. In the 1970s sweets producers in the Netherlands realized that they all supplied the same retailers. By pooling their logistics *i.e*., to globally optimize storage and transport, they managed to save costs without giving up competition in the market. This gave way to a business practice called *co-opetition* (Bengtsson and Kock, 2000), which stands short for (horizontal) collaboration between competitors. It has since then spread to many business sectors. The first experiment consisted in organizing a new *platform* with a core functionality *shared* between competitors, *i.e*., storage, inventory, and re-distribution to retail according to local needs for any brand. The platform was owned collectively and there was no competition in relation to its usage. Each producer had previously used a service provider, who supplied storage, inventory, and transport depending on what products of that brand retailers needed locally. Each service provider had operated out of its own *private logistics platform*. After co-opetition was introduced, they had to reorganize their ecosystems in order to survive with a reduced total income. Someone down the line was bound to be unhappy. Producers on their side didn't change how they organized production and competed in the market. Getting an advantage (reduced costs) out of change, without having to change the core business, is a strong incentive to switch platforms.

In the case of sector-wide authorized data transmission (authors' first case study), organisations were threatened by the emergence of a central service platform for smart- farming. They launched a counterproject that ended up building a digital interoperability platform for co-opetition (Sandoz, 2020). Both projects finally failed to scale up across the sector, because once organizations had reached their political objective of preventing the service platform to prevail, they dropped their shared concern for data management and fell back into doing business as usual.

If the interoperability platform had been widely adopted by organizations, a new question might have arisen: would their IT-service providers *pool* to co-opete in order to supply the new tools needed by their customers, or would they have resisted change, relying on their strategic position (Saadatmand *et al*. 2019)?

More importantly, the interoperability platform for authorized data transmission might have provided a mechanism to articulate the service platforms of the peer organizations into a broader, sector-wide, information infrastructure.

Traceability, on the other hand, is a form of collective control implemented by producers, transformers and distributors, regulators and consumers, etc. in value- or supply-chains. Shared concerns and requirements are *collaboratively* implemented in order to guarantee certain properties of objects.

The initial implementation of the animal traceability platform we studied (Stiefel and Sandoz, 2022) relied on a centrally positioned S*IS* that provided the consistent ordering of events and their transmission between event- producers and event-consumers. This position induced dependencies of client *IS*s towards the *SIS*. However, in large value chains like food production, most actors use only a small subset of all the types of events that are traced, and encounter only a small number of the events of those types that eventually occur. Technologies, modes of production, regulation, and products change constantly. The actors who are directly concerned by a change adapt quickly, whereas the others don't even see it. Providing traceability without imposing to the actors concerned any dependency towards actors that are not concerned, makes sense. Looking deeper into requirements for traceability leads to relax technical constraints like centralization that are not anchored in the analogical reality. It becomes then possible to design an interoperability platform (or a loosely coupled collection of interoperability platforms) for traceability that can scale up independently of the sector's overall complexity.

If a core shared functionality could scale up (*e.g*., in the number of peers for authorized data transmission, or in the number of participants and in the types of events for traceability), then an interoperability platform might end

up spilling over into a neighbouring sector (*e.g*., healthcare). The platform could then possibly become an *articulation* between the information infrastructures of *different* business sectors. Eventually, this is what the TCP/IP and HTTP platforms did when their basic core functionalities spilled out of their original business sector which was academia.

8. Conclusion: untying the knot

In this paper, we argue that the centralized, proprietary software-based platform model is only one type among several. Using the central component of the definition of a platform (Tiwana *et al*., *ibid*), *i.e*., the *core shared functionality*, we identify two other generic types of platforms, that we call *intermediation* and *interoperability*. We give examples of these alternative forms (*e.g*., *DNS*, *resp*. TCP/IP) to the *service* platform generally understood under this definition (*e.g*., Facebook, or large organizational platforms as in Hanseth and Bygstad, *ibid*). If service platforms seem to be exclusively built in the application layer of the Internet, *intermediation* and *interoperability* platforms populate all of its layers (internet, transport, and application).

The paper raises a series of questions that we enumerate in conclusion, as avenues for further work. *First*, there seems to be a relationship between how users of the shared functionality at the core of a platform interact, and its preferred architecture-governance (Hanseth and Rodon, 2020) and ownership configurations. Service platforms (no direct interaction) are *by definition* centralized- proprietary; intermediation platforms (used in support of interaction) are *by default* decentralized and non-, or possibly shared-proprietary; interoperability platforms (direct interaction) are *by construction* fully distributed and non-proprietary. It would be interesting to further investigate this relationship by putting it to the test of other case studies: if a relationship exists, of what order is it (historical contingency *vs*. practical necessity or strong compatibility)? Is it possible to change a platform's type all the while keeping its core shared functionality?

Second, we argue that interoperability platforms could be a basis for more open sectoral infrastructures which would not be the mere multiplication of proprietary platforms under the control of their respective private and/or public actors. This is in line with the new commons developed by the works of (Benkler, 2014; Boyle, 2002; and Lessig, 1998). This hypothesis also deserves to be tested by case studies. Are all cases of sectoral *II*s, based on traditional service platforms? And, if not, do intermediation platforms also play a role?

Third, we have shown that platforms of different types can have dependency relationships between them. For example, some service or intermediation platforms depend on interoperability platforms and interoperability platforms might depend on intermediation platforms. The question remains: what implications can be drawn from this observation? Here again, empirical studies, at the scale of interactions between platforms types, could shed light on this point.

Fourth, we opened a breach with our story of Dutch sweets producers in the 1970s, without going much further. It would be interesting here, however, to see how far the analogy between analogical and digital platforms might take us (in the line with the work done by Schafer *et al.,* 2021). But the effort might require distancing ourselves from the concept of platform and instead finding *cases of platforms* with which to work the analogy. Similar to what Nicolas Verdier (2007) did in the case of the horse post office, showing how technical network thinking was already at work in the 18th century, before the very concept of network appeared in the 19th.

Fifth, but not last, we sketched a socio-technical imaginary of high-level interoperability platforms (authorized transmission, traceability) that could spill over between neighbouring sectors and possibly become an articulation between their information infrastructures. Questions: is this imaginary possible for interoperability platforms only, or does it apply, for example, to traditional service platforms? In any case, would a platform- *articulated* cross-sectoral infrastructure be scalable and sustainable? And finally, is this only an imaginary, or are there cases of cross-sectoral information infrastructures that we could study?

The call is out.

References

Bengtsson, M., and Kock, S. (2000) "Coopetition" in Business Networks—to Cooperate and Compete Simultaneously, *Industrial Marketing Management*, *29*(5), 411–426.

Benkler, Y. (2014) Between Spanish huertas and the open road: a tale of two commons?. *Governing knowledge commons*, *69*.

Boyle, J. (2002) Fencing off ideas: Enclosure & the disappearance of the public domain. *Daedalus*, *131*(2), 13- 25.

Bucher, T. (2013) Objects of intense feeling: The case of the Twitter API. *Computational Culture*, Vol. 3.

DeNardis, L. (2009) *Protocol politics: The globalization of Internet governance*. Mit Press.

DeNardis, L. (2012) Hidden levers of Internet control: An infrastructure-based theory of Internet governance. *Information, Communication & Society*, *15*(5), 720-738.

Hanseth, O. and Bygstad, B. (2021) Managing IT in Large Organizations as Platform-Oriented Infrastructures. A Norwegian E-Health Case, Working Papers Series, Nielsen, P. (Ed.) Information Systems Group, Department of Informatics, University of Oslo.

Hanseth, O. and Modol, J. R. (2021) The dynamics of architecture-governance configurations: an assemblage theory approach. *Journal of the Association for Information Systems*, *22*(1), 5.

Hanseth, O., Monteiro, E., and Hatling, M. (1996) Developing information infrastructure: The tension between standardization and flexibility. *Science, Technology, & Human Values*, *21*(4), 407-426.

Helmond, A. (2015) The platformization of the web: Making web data platform ready. *Social media+ society*, *1*(2), 2056305115603080.

Kumar, V. (2018) Book Review: Platform Ecosystems. Aligning Architecture, Governance and Strategy, *Journal of Information Technology Case and Application Research, 20*(2).

Lessig, L. (1998) Keynote address: commons and code. *Fordham Intell. Prop. Media & Ent. LJ*, *9*, 405.

Scott, S. V. and Zachariadis, M. (2012) Origins and development of SWIFT, 1973–2009 *Business History*, *54*(3), 462-482.

Méadel, C. and Musiani, F. (2015) *Abécédaire des architectures distribuées*. Presses des Mines.

Merrill, K. (2016) Domains of Control: Governance *of* and *by* the Domain Name System. In *The turn to infrastructure in Internet governance* (pp. 89-106). New York: Palgrave Macmillan.

Mockapetris, P. (1983) *Domain names: Concepts and Facilities* (RFC 882) and *Implementation and specification* (RFC 883), 15 Sept. 2022 https://www.rfc-editor.org/rfc/rfc882 and https://www.rfc-editor.org/rfc/rfc883).

Musiani, F., Cogburn, D. L., DeNardis, L., and Levinson, N. S. (Eds.). (2016) *The turn to infrastructure in Internet governance*. New York: Palgrave Macmillan.

Musiani, F. (2016) Alternative Technologies as Alternative Institutions: The Case of the Domain Name System. In *The turn to infrastructure in Internet governance* (pp. 73-86). New York: Palgrave Macmillan.

Nielsen, P. and Aanestad, M. (2005) Infrastructuralization as design strategy: A case study of a content service platform for mobile phones in Norway. In *Proceedings of the 28th Information Systems Research Seminar in Scandinavia. Kristiansand.*

Nielsen, P. and Sæbø, J. I., (2016) Three strategies for functional architecting: cases from the health systems of developing countries. *Information Technology for Development*, *22*(1), 134-151.

Poppe, O., Sæbø, J., and Nielsen, P. (2014) Architecting in Large and Complex Information Infrastructures. In *Scandinavian Conference on Information Systems*, pp. 90-104. Springer.

Puschmann, C. and Burgess, J. (2013) The politics of Twitter data.

Ross, J. W., Weill, P. D., and Robertson, D. C. (2006) Enterprise Architecture as Strategy. Creating a Foundation for Business Execution. Harvard Business School Press.

Saadatmand, F., Lindgren, R., and Schultze, U. (2019) Configurations of platform organizations: Implications for complementor engagement. *Research Policy*, *48*(8).

Sæbø, J. I. and Poppe, O., (2015) Federated Architecting in West Africa. In *Proceedings of the 13th International Conference on Social Implications of Computers in Developing Countries, Negombo, Sri Lanka.*

Sandoz, A. (2020) Inter-operating Co-opeting Entities. A Peer- to-Peer Approach to Cooperation between Competitors http://www.thinkmind.org/index.php?view=article&articl eid=bustech_2020_1_20_90020.

Schafer, V., Balbi, G., Ribeiro, N., and Schwarzenegger, C. (2021) *Digital Roots: Historicizing Media and Communication Concepts of the Digital Age*. De Gruyter.

Schiper, A., Eggli, J., and Sandoz, A. (1989) A New Algorithm to Implement Causal Ordering. In *International Workshop on Distributed Algorithms* (pp. 219-232). Springer, Berlin, Heidelberg.

Stalder, F. (2010) Digital commons. The Human Economy: A Citizsen's Guide.

Stiefel, L. (in press) Les données du problème. Une plateforme numérique inadaptée à l'agriculture suisse. *Etudes rurales.*

Stiefel L. and Sandoz A. (2021) Une plateforme en pair-à-pair pour l'échange de données : l'émergence d'un commun numérique, *Terminal,* (130).

Stiefel, L. and Sandoz, A. (2022) Alternatives à la concentration : une analyse des relations de dépendance sur les plateformes numériques. In *Proceedings of the 31st AIMS Conference,* Annecy.

Tiwana, A., Konsynski, B., and Bush, A. (2010) Platform Evolution: Coevolution of Platform Architecture, Governance, and Environmental Dynamics (Research Commentary), *Information Systems Research 21*(4), 675–687.

Tiwana, A. (2014) *Platform Ecosystems. Aligning Architecture, Governance and Strategy*. Newnes.

Verdier, N. (2007) Le réseau technique est-il un impensé du XVIIIe siècle: le cas de la poste aux chevaux. *Flux*, *68*(2), 7-21.