Abstract—FLOSS generates both enthusiasm and concern. In this paper we focus on the Canadian public sector, how it could benefit from FLOSS solutions, emphasising the strong connection existing between public function and Free and Open Source models. At the same time we analyse some of the major legal concerns linked with FLOSS, to distinguish real issues from Fear, Uncertainty and Danger (FUD).

Keywords—licences; compatibility; derivative works; advantages for public sector;

I. INTRODUCTION

Free/Libre and Open Source Software (FLOSS, or FOSS) is not primarily about improving the bottom line, 'free' is as in Free Speech [1]. In order to give even more clarity to the meaning, the Spanish word adopted is libre and not gratuito. Often and at many levels FLOSS is considered as an option for adoption by provisioning departments only on the basis of the asserted cost-reductions that it may bring. The acronym is not always used properly, and sometimes is replaced by some other non-standard terminology. For example, the Canadian Government recently made a 'request for information' using the label 'No Charge Licensed Software' [2].

Although looking at the terminology could be seen as a minor issue, many FLOSS advantages may be found only in software that fits strictly within its definition, and not within acronyms and labels that could look similar at the eyes of a non-discerning reader, such as 'free-ware' or 'share-ware' or 'no charge software'. Here we look at the attributes of FLOSS that are linked with the legal requirements under which it is distributed, as well as the framework under which the software is developed. We focus on the public sector advantages, i.e. government, public administrations, federal or provincial agencies and similar. The private sector may have other concerns that are not addressed here. The following sections address: basic terminology; real and presumed problems connected with licence compatibility, with some example of how important it is in legal related phenomena to call things by their precise names in order to avoid further misunderstandings; various beneficial aspects of FLOSS adoption by governments and public agencies; and a final section with relevant data from a recent research project along with a recommendation.

II. TERMINOLOGY

So what is FLOSS? Two main concepts of the nomenclature should be separated: Free/Libre and Open. There has been vexatious dispute between these two ideologies, which lie behind similar code development methods, that has cooled off over the last couple of years. "[T]he obvious meaning for the expression 'open source software' is 'You can look at the source code,' and most people seem to think that's what it means. That is a much weaker criterion than free software, and much weaker than the official definition of open source. It includes many programs that are neither free nor open source. Since that obvious meaning for 'open source' is not the meaning that its advocates intend, the result is that most people misunderstand the term' for the Free point of view [3]. Contra the Open Source Initiative (OSI) position: "it was decided at a conference that it was time to drop the moralising and confrontational attitude that had been associated with 'free software' in the past, and sell the idea strictly on the same pragmatic, business-case grounds that had motivated Netscape. They brainstormed about tactics and a new label, 'Open source', contributed by Chris Peterson, was the best thing they came up with" [4].

Despite advocates' different points of view towards one or the other position, and the different weight that each position gives to ethical and business concerns, the difference lies more in a philosophical level of abstraction than in a substantial one. This does not mean that there are no practical differences between these two definitions: the Open Source definition is some how broader than the Free Software one. However, much more than definitions, a provisioning department at the moment of deciding what type of software should be used must look to the legal requirements that establish the terms of use of the software, i.e. the licensing.

A. Licences

The first and most famous of these agreements is the GNU General Public License, (GNU GPL or simply GPL). The first version, February 1989, was followed in June 1991, by the second version. The wording "version 1 (or 2, or 3) or any later version" is often used. The current
version 3 dates back to June 2007. Between 50% and 70% of all FLOSS projects are released compliant with one of the versions of the GPL [5]. Although the GPL covers more than a half of the major FLOSS projects there is a plethora of other licences that are commonly used. The Open Source Initiative (OSI) reports 55 OSI approved licences. The Free Software Foundation (FSF) reports 43 Free Software approved licences compatible with the GPL and 39 licences that are deemed Free Software but not GPL compatible.

In many cases, to be GPL compatible is a matter of version, and different versions of the same licence can be either GPL compatible or not. It is interesting to note how, technically speaking, version 2 and version 3 of the GPL are not compatible. The issue of compatibility is of paramount importance when a public agency decides to develop its own software tools, either because it must be aware of the licensing already set up for the existing code, or if they want to use or further develop tools that are publicly available. They have to know what they are allowed to do with them, and to what extent, if any, they can be merged and under which conditions.

B. Updating GPL

While GPLv3 aims to maintain and further the principles of GPLv2, technological pace and new threats to FLOSS obliged GPLv3 drafters to insert some provisions to address the current issues. The drafting of v3 was a socially distributed effort, and many criticisms that emerged on drafts regarding new requirements have found their place in the current version. In particular, besides a general improvement in terminology, particularly for internationalisation [7], and a more detailed section on definitions we can see the following new sections: Digital Rights (or Restrictions) Management/TiVoization (sec. 3); Licensing Patents (sec. 11); short term compliance (30 or 60 days) before automatic termination of the licence (sec. 8); clarification regarding peer-to-peer distribution of object and source code (sec. 9); and a less monolithic general framework (sec. 7).

Unfortunately, since both v2 and v3 are strong copyleft licences (see infra), this excludes their mutual compatibility. If a work is released under GPLv2 and somebody wants to distribute a modified version of this work, such modified version will be under GPLv2. The same happens with GPLv3. The problem arises when a modified version is based on more than one program, where at least one is under v2 and the other under v3; in a case like this, there is no legal way to make them compliant.

However, it must be kept in mind that the copyleft requirement applies only to modified versions, or for versions based on a previous work. That is to say, it is possible to distribute a software package (say an operating system) containing programs under different licences, eventually even not compatible with each other. This is what happens in the most common GNU-Linux distributions, where the Linux kernel distributed under GPLv2 happily coexists with other tools, or applications released under GPLv3 (or many other non compatible licences). Cases like proprietary Loadable Kernel Modules (LKMs) or binary blobs i.e. those object files loaded into the kernel without publicly available source code are considered borderline cases, meaning that in some limited circumstances their use is accepted because it is recognised that they do not form a derivative work of the kernel.

GPLv3 may be combined with important licences that are not compatible with the former version, namely the XFree86 (v. 1.1), the Apache (v. 2.0) and the GNU Affero GPL (v.3) licences. In particular, the latter is a GPLv3 licence specially aimed for network-interactive software, thus allowing users of web-applications to be able to receive the source code (technically speaking to run a server is not an act of distribution).

C. BSD licence and versions

A criticism of FLOSS licence regimes is as to the naming system. Law requires certainty in many aspects, including terminology. If versions in regards to the GPL licence sounds confusing, than even more confusion exists with compatibility issues regarding the Berkeley Software Distribution (BSD) licence.

BSD is a FLOSS licence (FSF recognises it as Free Software) but it is a permissive licence (meaning neither strong nor weak copyleft, see infra). The BSD licence should, more correctly, be referred to as an entire family of licences, rather than to only one. The main reason for this classification is the multiple modifications that the original licence has suffered, thus when software is distributed with a BSD licence, it is of pivotal importance to know exactly which version.

The new, or revised, or again 3-clause BSD licence is, clearly, Free Software and also GPL-compatible. However, this compatibility does not exist when referring to the original, or old, or 4-clause BSD licence. In the latter, an extra clause (n.3 in the text’s body) imposed a requirement which made it incompatible with GPL. This clause, also called the ’advertising clause’, required authors of derivative works to include an acknowledgement of the original source, which, making it short, could lead, and sometimes has, to many pages of acknowledgements. Each of these sets is basically composed of the same licence with some slight variations in the wording.

In addition to these two main categories, the BSD family has grown luxuriantly. Among the more widespread, there exists the NetBSD, the 2-clauseBSD (similar to the MIT), the FreeBSD, and the Clear BSD. All of these variations of BSD are usually GPL compatible, though this does not mean that their actual wording should be ignored. On the contrary, it is important to know, for instance, if the licensor
reserves the right to sue you for patent infringement or not (see the Clear BSD).

III. COPYLEFT’S REACH

In certain production circumstances the use of some types of FLOSS licence is perceived to be problematic. Some that create and distribute ‘packaged’ software have detracted from GPL due to its so-called viral nature. The word viral is not fortunate, as it projects a negative connotation upon a clause in a legal document. Seeing such characteristic with favour or not is a matter of personal choice, but as a matter of legal definitions, it should be referred to with a more neutral epithet: here we will refer to this characteristic as ‘persistance’. One of the characteristics of the GPL is its strong copyleft status. Strong copyleft licences are those licences that require any subsequent distribution of the work or a modified version of the work, must be under the same licence. A new program based on GPL licensed code must be distributed under GPL. This persistency has represented a major issue in the field of FLOSS. Some supporters of FLOSS models that are based on non-persistent licences, but permissive licences (i.e. like the BSD), have accused the GPL of cannibalising BSD software, insomuch as the permissiveness of BSD-like licences permits protected code to fall under the GPL, while the other way around is not possible due to the copyleft requirement of the GPL. GPL supporters argue that it is not the GPL cannibalising code, but rather, it is BSD that permits every type of licence and even propertisation of BSD software. Copyleft proponents say, is necessary to protect and foster the development of a contributory commons.

There are some FLOSS licences that are copyleft but their requirements are not as strong as the GPL. Consequently, they are labeled ‘weak copyleft’ licences. Examples of this category are the LGPL (where L stands for Lesser), the Common Public Licence, and the Mozilla Public Licence. These licences allow combining the software with other types of licensed software without the necessity of distribution under the same licence (but this does not mean that they don’t need to be compatible: the CPL and the MPL, differently from the LGPL, are not GPL-compatible [16][17].)

The difference between weak copyleft and permissive regimes is that the possibility to combine, say, LGPL and closed-source software without turning the output into LGPL is available only for linking activities. If a piece of software released under the LGPL is going to be modified in order to create a derivative work (a new version, or a fork) the new software will have to be released under the same licence (or eventually the standard GPL), thereby fulfilling the copyleft part of the label. Since persistency only works for some types of activities and not others (linking in LGPL case), such copyleft is not strong, but weak.

This intentionally brief overview of the compatibility issues regarding weak copyleft licences necessarily brings us to the concept of derivative work. The aim of this paper could not be an exhaustive analysis of what this concept could legally mean, since due to the ubiquitous nature of the Internet, such a survey would furthermore have to be completed for all jurisdictions. What we can analyse is the meaning of derivative works in the case of a program being linked by another one, usually a library, and observing the unique consequences deriving from the wording of the GPL in cases of dynamic linking and static linking, and ultimately whether this distinction does, or should, matter. A program statically linked with a library, creates with the latter a new, modified work. If either piece of software is released under the GPL, the derived work (the program statically linked with the library) shall be under GPL. Since part (a substantial part) of the library is copied into the executable of the program at compile time, the output is the program plus the library (substantial part thereof), and thus a new work based on the two precedents. If one of the two works is released under the GPL, the new derivative work will have to be under the same licence because the GPL requires that any derivative work be under the same licence.

A more complicated case is that of a program that is dynamically linked with a library. In such a case, no substantial part of the library is present into the executable, so besides being connected, the latter is not a derivative work. However, while the FSF and GNU agree with this general framework, they further affirm that when a dynamically linked library and program share a more ‘intimate’ existence, they should be considered once again a derivative work. More precisely “[i]f the program dynamically links plug-ins, and they make function calls to each other and share data structures, we believe they form a single program, which must be treated as an extension of both the main program and the plug-ins, while if the program uses fork and exec to invoke plug-ins, then the plug-ins are separate programs, so the license for the main program makes no requirements for them”. [11]. A complex, borderline case, where in presence of a dynamic linking structure the FSF and GNU support the thesis of a derivative work (extension of the two codes), due to the relation between the two pieces of software, which is so strict (reciprocal function calls, sharing of data structures) that, even in the absence of a substantial portion of the source code of one of the programs into the other, the functional result is not far from it. Nonetheless, FSF and GNU recognise the presence of undefined areas "If the program dynamically links plug-ins, but the communication between them is limited to invoking the main’ function of the plug-in with some options and waiting for it to return, that is a borderline case" [11].

The Canadian Copyright Act gives little guidance for such situations, only generally reserving the right to "produce or reproduce the work or any substantial part thereof in any
material form whatever[...]and to authorize any such acts” to the rights holder, and adds some specific cases of adaptation, that are inapplicable to software (sec. 3, especially d. and e.). The Act is in good company with its southern neighbour, as well as many European countries, as the legislation does not reach such fine granularity to deal with a library dynamically linked to a program with which it shares system calls and data structures. The courts have not yet had a chance to address this issue. However, it must be noted, in Canada, while rewriting a computer program from one language into another could be interpreted as a translation under certain circumstances [12], compiling the source code into object code is an act of reproduction [13]: the main consequence of this distinction, is that to compile a program (being either an application or library) requires the mere right to reproduce [14].

Determining exactly what a derivative work is within linked computer programs, is a contentious issue. It obviously depends on the legal system where one claims protection. However, there are claims that the issue of static and dynamic linking is a red herring, what really matters is not the name of a program or call (mkisofs, ld, exec, or the like), which undoubtedly has functional consequences, but the grade of dependency or independence between the two programs. This relationship establishes whether the output is a derivative work or a mere aggregation [15]. This latter approach introduces some uncertainty in the sense that it suggests a case-by-case analysis, rather than a "static = derivative" equation. The door is still open to deeper analysis on this issue, and also within the same FLOSS community, one of the fathers of Linux (Linus Torvalds) once affirmed that ‘there was not much need for the LGPL’ [15].

IV. NOT JUST MONEY

We have looked at some of the most debated issues surrounding the FLOSS option, and have seen that while some of them are better characterised as technical interpretations rather than real problems, there are others that represent complex situations requiring specialised legal and technical advice in order to be adequately addressed. We now look at the benefits connected to FLOSS.

Obtaining FLOSS requires nothing more than an internet connection. Inherent in both the FSF and OSI models is the ability for anyone to access the code. There are no royalties to be paid, no required tie-in to service contracts, and no up-front acquisition costs. Besides the economic aspect, there are many advantages of adopting FLOSS. Price is not the primary advantage, even though sometimes price is erroneously taken as the main point, with the consequence that FLOSS is assimilated with other non-immediate-fee software. Such naivety should be avoided, especially when the interested entity is a public body, as the main objective of its activity is not to make profit, but rather to offer public benefit. A most important aspect of FLOSS is the availability of the source code, and the ability to modify and redistribute improvements is a contractual obligation existing in all its licences, which entails legal, technical and economic consequences.

Accountability and transparency Source code availability permits users to know in depth what the programs does. Without the source code one can only deduce what the program does through expensive and time consuming reverse-engineering without ever having the opportunity to know all of the original code. Source code availability is critically important for software applications in the core areas of government (such as national defence and homeland security, financial and economic administration, and health databases), as well as the fundamental infrastructure of public administration [6]. The possibility for the general public to understand and to rely on the activities of public bodies is directly connected to the use of a software model that is transparent and accountable, which is an essential element of e-Democracy. This is a fundamental aspect in providing citizens with the guaranties of fair, efficient and impartial administration of the public good. These are requirements that all the public bodies should conform with. An example of this can be seen in electronic voting systems [8].

Interoperability. The availability of source code allows for better interoperability with other applications. If an application is not perfectly compatible, the availability of the source code, combined with the contractual permission to use and adapt it, allows for work on the code with interoperability as a result. If there were FLOSS and closed source software (proprietary), greater compatibility is possible than what would be achievable in the case of two closed source software provided by two different suppliers [10]. This is of particular interest for public bodies since it grants the possibility to share resources among many different departments and agencies. Despite being autonomously organised, public bodies do not suffer from the strict competition that affects corporate entities, what allows for strong scale economies with significant savings for the whole public administration, and consequently for tax payers. In some jurisdictions (eg Italy) this is prescribed by law [9].

Avoid lock-in Vendor lock-in is the phenomenon that causes customer dependency on a vendor with regard to a good or service. Switching vendors has high transactions costs connected with technological and organisational changes, and in some cases penalty clauses due to early cancellation of a supply contract. These switching costs are pernicious to the market and can represent strong barriers to entry. With closed source software the customer is generally bound to a specific supplier, both contractually and technologically. As an example, in the case of free ware, one typical business model sought is to create lock-in. In this case, once the lock-in has occurred the software distribution model can switch to a traditional priced one since the transaction costs connected with the migration
to another type of software are prohibitive [28]. In the case of FLOSS, both the licence and the technology allow for a supplier-independent business model [18]. For public administrations, it is mandatory to choose suppliers that are able to grant reliable services, at good prices, and provide for long-term maintainability (usually public administrations last longer than private companies). However, it is also critically important that at a given point, if a better offer or player enters the market, that there shouldn’t be any tie impeding the public body’s transition to the more efficient solution. This will immediately reflects on the cost and quality of service enjoyed by citizens.

**Long-term maintainability** FLOSS implies Open Formats. Open Formats are those formats that are publicly documented so as to permit anyone to implement programs (both FLOSS and closed source software) that can optimally use, store, and retrieve such data. This is another manifestation of the absence of any lock-in problems [19]. Many times the reason for remaining bound to an old supplier is that they are the only one owning the (closed) format technology enabling data retrieval.

**Technological ecumenism** A public administration cannot discriminate the public based on the type of software used. A private company has the option to choose to use closed source software compatible with 85% of the software used by citizens, and incompatible with remaining the 15%: the market will decide if this decision pays. A public administration cannot exclude 15% because they chose a different type of software. FLOSS is the solution that grants the highest compatibility, thus minimising the phenomenon of technological exclusion by both FLOSS users and closed source software users.

**Security and error correction** Security is not a static concept that can be reached and once reached easily maintained. FLOSS is known not only for the transparency and accountability of its code, but also for its stability and intrinsically greater security. It is a common principle in computer science that the security of a system depends on the quality of its structure, and not on its obscurity (a variation of the Kerckhoffs’ principle in cryptography). Only if the source code is available is there the possibility for quick bug-correction and exploit-detection. In the case of FLOSS, the pace at which the stability level of the code grows is much faster than in other types of software, where it is necessary to wait for the supplier security updates [20].

**Democracy and pluralism** FLOSS in the public sector is more generally a matter of democracy [21]. In case studies such as electronic voting machines, or “technology-enhanced” trials, the people need to rely and trust not only in their representatives and the courts, but also the process of electing the candidates or of condemning the guilty [21]. FLOSS seems to epitomises those basic principles so commonly found in many constitutional and fundamental charters, of fair administration of the justice, of pluralism, of freedom of expression, and of access to information and knowledge.

**Portability to other languages** The possibility, both technical and legal/contractual, to translate software into any language is of paramount interest if due importance is given to linguistic and cultural pluralism. Although this could sound as a more European, Asian or African based argument, also in America (both North and South) language plays a key role in the protection of indigenous and traditional knowledge. This feature of FLOSS may be easily confirmed by checking the language packs or language ports of some of the most widespread projects and comparing them with similar non-FLOSS products. For example, Firefox v3.1 has 62 language ports and 78 different languages packs, among which many minority languages are present. On the contrary, Internet Explorer 8 has 43 language selection possibilities. Opera has 41 supported languages, Chrome 44, Safari 18. The difference is even greater in the case of the office suite: OpenOffice.org has 123 supported languages, while Microsoft Office has 35. Finally Outlook 2007 is available in 14 languages, while Thunderbird 2 in 39. The reason for this difference in language policy resides clearly not only in the sensibility of the project managers but rather in declared legal, contractual and technological features of FLOSS. It also lies in the transaction costs associated with the development of these tools, which once again has beneficial consequences for competition on the market and the technological development.

**Fostering competition** Another major advantage of FLOSS is that it creates and favours a more competitive ICT environment usually populated by many local Small and Medium Enterprises. Licence fees, from a microeconomic point of view, represent huge barriers to entry to market, thus favouring monopolistic and oligopolistic situations. As it has been reported [22] [23], a public administration investing in FLOSS solutions is usually interested in hiring or contracting with local ICT companies for services like updating, maintenance, training, and customisation. In this way the immediate benefit on local economies is apparent.

**Total Cost of Ownership** A major saving in using FLOSS is in terms of royalties as there simply are none. Which per se, represents a huge amount of money. Due to the alumni effect, a lot of people learned how to use computers through non-FLOSS applications, which means that even though FLOSS solutions are user-friendly, there are still some costs connected with migration. There are savings that are not strictly connected with licences. Some of them have been already identified (no lock-in, enhanced security, etc), while others are more concealed (such as cross-platform availability, maintenance, updating and long-term upgrading, compatibility with ‘older’ hardware, etc) [24]. Taking into account all these variables provides a better portrait of the actual benefit in terms of economic and financial costs. From many studies, it is undisputed the huge Total Cost of
Ownership savings resulting from the use of FLOSS. With respect to the public sector reports show yearly savings in the billions of dollars [25]. One further benefit is that the agency taking the FLOSS route will need to spend on the development of internal staff skills, which means that the skill base for the organisation is improved, giving better overall support.

V. DATA, RECOMMENDATIONS AND CONCLUSIONS

Looking at these considerations, FLOSS gives the best solution for public administrations. At least it should be given priority in all those cases where an alternative to purchasing proprietary software can be found with a FLOSS solution. A long list of public administrations around the world have already started, or are seriously considering, migrating from proprietary to open code. Among the most successful initiatives includes "LiMux the Free Software in Munich" project. They report to already have 1,200 workstations migrated to Gnu-Linux, 12,000 using Open Office, and the 100 % of the city administrations using Firefox and Thunderbird [26]. Another interesting case study can be seen in Spain, where the different comunidades autónomas (the Spanish provinces) have different levels of FLOSS implementation, all coordinated by a public agency: Centro Nacional de Referencia de Aplicación de las TIC basadas en fuentes abiertas.

All these advantages are connected with FLOSS, that is, with software that complies with either the Free Software or with the Open Source Software definitions, if not with both. The "dollar price", connected with the absence of royalties, is but one of the potential savings: "Contrary to what is often assumed, cutting costs was not the main reason for the migration. The motivation is independence [...] now we’re able to decide on our own how we want to spend our IT budget in the long run [...]" [22]. There are many other advantages that have stronger economical value, both in the short and in the long run, that are only present in the case of FLOSS and with the technical and legal availability of the source code and the possibility of its modification and redistribution. In addition to the monetary savings connected with the absence of licence fees, there are huge advantages that relate to independence from software providers, the creation of a competitive market usually on a provincial, regional or national level, transparency and accountability, the ease of customisation, digital inclusion and pluralism, and the further savings connected with scale economies amongst different public administrations; this confirmed, amongst others, by the Spanish National Observatory on FLOSS [27].

In 2008 we started a project intended to map the diffusion and awareness of FLOSS in the Canadian Public Sector. Preliminary data and analysis show some interesting early observations. Among the 53 government bodies and agencies we contacted, 20 participated, and of them 14 said they use some FLOSS, but only partially and mainly on servers. Usually this is connected with particular interest by the IT staff, while the department head did not forbid it, but there is no official support or endorsement. Two departments noted that they use FLOSS and that they are somehow encouraged in doing so due to the possible advantages connected with the use of FLOSS into the public sector. Six departments declared that they do not use any kind of FLOSS, two of which indicated that that they did not use FLOSS "because we have only original, regularly licensed software. We pay for our licences". On very large federal department responded that they have official internal directions not to use any FLOSS and that they had to remove 'small amusement machines' from their IT departments. This is still a partial data set and it requires further elaboration before being adequately interpreted. However, one thing can be already inferred: any policy that the Canadian government may have for FLOSS in its departments is not implemented effectively.

In this paper we have looked at some aspects of the state of the current debate regarding the adoption of FLOSS by governments and other public bodies and have tried to clarify some of the more complex legal issues. In the past, the transition of a society into a real, participative democracy has largely depended upon the need for the people living in that society to learn how to speak, write, and communicate amongst themselves and with the institutions that they elect and that represent them. This is fundamental in order to be real participants of any social organisation. The parallels in the digital arena are not that absurd. By concealing code and hiding how software works, we are creating a new form of illiteracy that will take a long time and a huge amount of money (on the top of other disruptive socio-economic consequences) to overcome. That is the main reason why, even above the cost-saving argument which has its own appeal, the Canadian Government should take a real definitive step toward the migration to FLOSS solutions, and not only to solutions that do not charge licence fees.

As a final observation, during fiscal year 2005-2006 Canada spent over $425 Million on software licence fees.

ACKNOWLEDGMENT

The authors would like to thank the Social Science and Humanities Research Council and the Law Foundation of Ontario for their support.

REFERENCES


[2] See http://www.merx.com. All the websites cited in this work have been last visited during August-September 2009.


[9] See arts. 68 and 69 Codice Amministrazione Digitale, Legislative Decree 7 March 2006, n. 82, (as amended).


[22] C. Gerloff, ‘Declaration of Independence: the LiMux project in Munich’, Open Source Observatory and Repository (OSOR), European Commission’s IDABC project.


