Open Source Software: Risk or Opportunity?

Sylvain Wallez

Open Source Software: Risk or Opportunity?

Sylvain Wallez

Anyware Technologies – http://www.anyware-tech.com/ – sylvain.wallez@anyware-tech.com
Apache Software Foundation – http://www.apache.org/ – sylvain@apache.org

Abstract: Open source software has changed large parts of the software industry landscape in a few years period, and is now emerging in the embedded software domain.

After a definition of open source software, this paper will explain why industrials use it and why software publishers cannot ignore this evolution. We will also discuss if open source software is a risk or opportunity for software vendors.

In the light of what happened in the information systems and middleware domain, we will see what trends are likely to change the embedded software industry in the coming years.

Keywords: open source, business model, tools, software vendors

1. Open source software, its origins and its achievements

Open source software is a generic term that encompasses many different realities, from student experiments to large industry-backed products that power the Internet's infrastructure, revolutionizes the desktop and reshape the development tools marketplace.

2.1 Origins of open source software

Free software: In 1984, Richard Stallman, a researcher at the MIT, started the GNU project. The project's goal was, simply put, to make it so that no one would ever have to pay for software. Stallman launched the GNU project because essentially he felt that the knowledge that constitutes a running program – its source code – should be free. If it were not, a very few, very powerful people would dominate computing.

Contrarily to commercial software vendors that saw trade secrets that must be tightly protected, Stallman saw scientific knowledge that must be shared and distributed for innovation to continue.

To avoid companies to reuse the public domain code for their own profitability, Stallman set up the GNU General Public License (GPL). It basically says that you may copy and distribute the software licensed under the GPL at will, provided you do not inhibit others from doing the same. The GPL also requires works derived from work licensed under the GPL to be licensed under the GPL as well, thus forbidding commercial proprietary use (and abuse) of free software.

The word "free" has two meanings in English: "liberty" and "at no cost", which is often explained as "free as free speech" vs. "free as free beer". Stallman was referring to the free speech. This political message combined with the constraints of the GPL led many software companies to reject free software outright, as their primary goal is to make money, and not the philanthropic action of adding to the common knowledge.

Open source software: in 1997 a group of leaders in the free software community tried to find a way to promote the ideas surrounding free software to people who had formerly shunned the concept. They were concerned that the Free Software Foundation's (GNU's umbrella organization) anti-business message was keeping the world at large from really be involved in free software.

One of the outcomes of the discussions was that some marketing was needed to win mind share and move away from the political meaning of "free" used by the FSF. This is where the term "open source" comes from (also written as OSS – Open Source Software). A series of guidelines were crafted to describe software that qualified as open source.

The Open Source Definition [1] allows greater liberties with licensing than the GPL does. In particular, the Open Source Definition allows greater promiscuity when mixing proprietary and open-source software.

Consequently, an open source license could conceivably allow the use and redistribution of open source software without compensation or even credit. This allows companies to use open source software in their proprietary products, but also allows them to release some of their software as open source.

But why would a companies release their source code for free to the world, including their competitors? For a number of reasons, but the most

compelling is that it gets greater market share for their code. In this way, giving away source code is a very good way to build a platform. We will largely expand on this subject in the paper.

Nowadays, along with "Free Software" and "Open Source Software", the "FLOSS" acronym (Free/Libre and Open Source Software) is used to refer to both types of software whose source code is available publicly.

2.3 Achievements of open source software

FLOSS is something that cannot be ignored, as everybody that uses a computer nowadays uses open source software directly or indirectly.

The internet infrastructure: the GNU project's original goal was to build a free version of Unix at a time where the Internet was growing, and the first major achievements are in the server infrastructure: the Internet's domain name server infrastructure is powered by BIND, and most email messages are routed either by Sendmail or QMail

Separate from the GNU project, and leader of the OSS movement, the Apache server powers more than 2/3 of the web servers worldwide, and most of these servers are running a FLOSS variant of Unix, such as Linux, FreeBSD or OpenBSD.

Development tools: next to infrastructure servers came development tools. It started with Emacs, a versatile text editor that is so extensible that full-fledged IDEs were built on top of it.

A very important project is GCC, the GNU C Compiler that over the years has become a generic purpose compiler for many languages and many target hardware platforms, and is the base of GNAT, a well-known Ada compiler.

More recently, the IDE market has seen the emergence of serious open source contenders. This started with NetBeans, a Java development environment. Then in 2001, IBM decided to open-source a large part of their commercial IDE, and created the Eclipse project. This project is now developed by an impressive number of companies that contribute resources for free, and which led Eclipse to now be the platform used by most of the large-scale IDEs on the market.

Desktop software: this is a domain where FLOSS started rather recently, but where it is quickly growing. This includes tools such as the Mozilla web browser and email clients, the OpenOffice suite that is more and more chosen in public sectors and administrations. Also, the Linux graphical environments, Gnome and KDE have now reached a quality that allow the use of Linux as the desktop operating system in many enterprises, along with being the operating system of choice of many hobbyists and students. This last category is worth considering seriously as they are tomorrow's engineers and managers.

2.4 Open source licences

One of the first things to consider when it comes to open source software is the licence. There is a wide variety of licences that are approved by the Open Source Initiative [2] that specify what you can do with the software and the associated requirements you must comply with.

These licences can be classified by the constraints they impose to commercial users.

The GPL (General Public Licence): the original "free software" licence. This is the most constraining, as it states that GPL-licensed software must be redistributed in source form, and that any work derived (i.e. built with) GPL'ed code must also be licensed under the GPL. This is why this licence is said to be "viral". Practically, this means that no closed proprietary products can be built with GPL-licensed software.

There are some ways to build commercial value with the GPL though:

- By using GPL-licensed code but not redistributing it. This includes in-house use and also selling services with the software. A number of large e-commerce websites heavily use GPL'ed software.
- By using "dual-licensing": a company that owns the copyright of a product can distribute it under the GPL for those people that accept its constraints, and using a traditional commercial licence for commercial users. This is a way for companies to win market and mind share with the free version while still making revenue with the commercial version.

  This dual-licensing model is used for example by MySQL with the namesake database engine or AdaCore with GNAT, the GNU Ada Translator.

The LGPL (Lesser General Public Licence): this is a relaxed version of the GPL, which restricts its viral nature to "extensions" of the software. LGPL-licensed code can be used in proprietary software.

However, this license has a clause that states that users should be able to upgrade LGPL-licensed part of the software on their own, and that it may include the ability to reverse-engineer the product.

The MPL (Mozilla Public License) and the EPL (Eclipse Public Licence): these licences were written by two companies, Netscape and IBM respectively, to cover the large code bases they open-sourced.

These licences put no constraints on derived works, which can choose any license they like, including closed-source commercial ones. It does require though, that any modification of the original source code be republished to the originating project.

The Apache and BSD licenses: these licenses allow unlimited use and modification or the open source code base, but require crediting the original author in derived works.

These licenses are very commercial-friendly and projects using this license often receive some corporate investments.

2.5 Intellectual property and patent issues

An important point to consider when using open source software is the patent issue. Many licences have some clauses protecting users from being sued for patent infringement because of their use of software covered by that licence.

The Apache Licence, for example, states that each contributor to the licensed software grants it users a perpetual, worldwide and royalty free patent license to use the software.

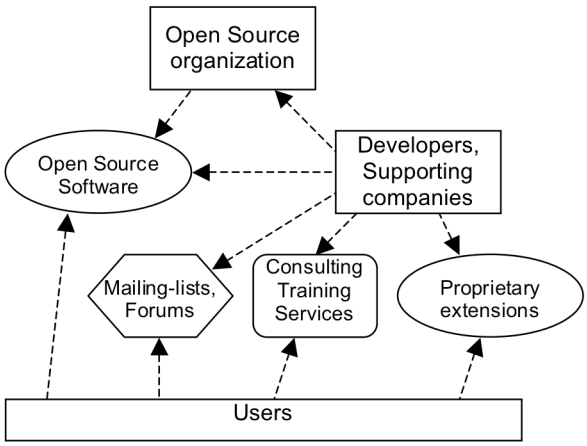
To allow this clause to be effective, most open source groups and organizations require that, prior to contributing code, contributors do sign a "contributor licence agreement". Basically, this agreement states that the contributor has the authority to decide what should be contributed as open source, and accepts the contribution to be redistributed under the open source license. And when the contributions are developed on paid time, the employer also has to sign a "corporate contributor license agreement".

It is to be noted that contributing code to an open source organization doesn't mean giving away authorship and copyright, but just a license to use the code. The original authors keep the copyright on their work (in some countries such as France, giving its copyright is actually legally not possible) and thus the right to do whatever they want with it.

3. Open Source ecosystems

An open source product cannot exist without a community around it. This community not only takes care of the product itself, but also builds a number of peripheral activities that derive from the open source product and nurture both its development and its developers.

The parties involved and the process described here below is what can be found at the Apache Software Foundation [3], an organisation that is not only well-known for the Apache web server, but also for being one of the first and most successful open source development groups.



Open source software: this is the result of the common work of all members of the ecosystem. Good software is not all that is needed to have a successful project. A motto of the Apache Software Foundation is that "the community is more important than the code", meaning that without a surrounding healthy group of people, a software product, as good as it can be, is a dead end. Tim O'Reilly once analysed some distinguishing features of successful open source projects, and came to the conclusion that architecture played an important role in allowing people to contribute. This is the "architecture or participation" [4]: the product must allow people to quickly understand pieces of it and start contributing on peripheral parts without having to know the inner details of the whole product. So, a componentized and decoupled architecture, along with being a good technical practice, is also a key community growth factor.

Users: they are obviously needed. Without users, a project is dead. They first download and use the product, then start asking questions on discussion forums. Some will also start contributing either by answering on the forums or by sending code corrections and evolutions.

Users will also often ask for support that goes beyond what the forums can provide, and then will enter a traditional business relation with field experts, most often employed by sponsoring companies. They also will be interested by commercial extensions to the open source software, that allow them to achieve their project's goal and deadlines faster while still benefiting of the advantages of open source foundations for their project.

Mailing-lists and Forums: open source projects are developed in the open by teams that are geographically distributed. The mailing lists and forums are the virtual places where the projects live.

There are often several discussion areas, dedicated to users and developers.

The users forum is where people can ask questions and get answers, which are given both by the developers and other users. Compared to commercial vendors support offerings, users forum are free of charge, and allow people to get in touch directly with the developers.

However, the users forums do have some limits because of their free-of-charge nature. Questions must be concise and precise, otherwise analysing the problem takes too much time for a volunteer effort. What that means is that these forums require people to have the necessary skills to spot their problems, and that questions of a more general nature, such as architectural concerns cannot be answered there. This is where sponsoring and expert companies come in the game.

Developer forums are where actual development of the open source product takes place. Design discussion happen in public, which allows users to know what happens, voice their concerns and bring their ideas, participate and ultimately, after a number of contributions, become part of the development team.

Development team: also known as "committers" because they have the rights to "commit" modifications to the project's configuration management system, they are those who write the code of the open source product. Most open source projects use a meritocratic process. The more people participate, the more they earn merit and thus rights. Developers are grown-up users that have contributed on a regular basis and were therefore invited to join the developer group.

Supporting companies: licences have an influence on the nature of the development group: commercial-friendly licenses allow companies to have some of their employees working on open source products. This is not a philanthropic act, but a business strategy, as we'll see later, since it allows companies both to have an influence on the product's roadmap and be in a privileged position to sell derived products and services.

In this regard, developers are for the company much more than technicians: they are also strong marketing assets and the link between users of the open source product and the company's associated offerings.

Open source organizations: last but not least, the organization that hosts the project. Users too often see this important actor in the system as a simple web site. A successful project cannot exist without some kind of structure, and over time some large organizations have emerged, each specializing in its own domain.

Open source organizations have several roles with regard to the other elements of the ecosystem:

- Infrastructure: a project needs a website, a configuration management system, some mailing lists and forums. The hosting organization provides hardware resources and network bandwidth to the project. This is the bare minimum that can be offered by the organization.
- Legal framework: contributing and using some open source code inevitably leads to the question of patent issues. What if I accidentally violate a patent with the code I contribute? What about users that use that code? The role of the organization is also to protect users and contributors from legal issues by taking responsibility of them.
- Oversight and guidance: as in all collective human activities, personal problems and conflicts can arise in a project. And since the various contributors often have no authority on each other, the role of the organisation is to take appropriate actions when mediation has failed. In some rare occasions, this can go up to evicting some people from the development team.
- Incubation: working on an open source project doesn't follow the same rules are traditional in-house development. Large organisations such as the Apache Software Foundation or the Eclipse Foundation have set up incubators [5] where newborn or open-sourced projects start their life and learn "the open source way".

4. Why industrials use open source software

About six years ago, the author's company started an offering related to web application development in the information systems (IS) area. This offering was built using a number of open source products, but we avoided mentioning "open source" when showcasing our products to customers. At that time, it conveyed a negative image of low-quality code, written by insomniac students or academic researchers. Today things have dramatically changed, and stating that our offering is based on well-known open source products, the development of which we're actively involved in, is considered as a real bonus by our customers.

This isn't a temporary fashion effect, but a real long-term trend. Today, most of the infrastructure and middleware software needed to build a project in the IS domain are available for free. This is a side effect of the standardization of protocols and service

interfaces that has happened over the last years in this domain.

The growing complexity of IS systems had a number of consequences. Rewriting everything from scratch for each project wasn't economically viable, and components had to be reused to speed up development. Also, projects had to be an assembly of products from various origins because no company, except huge ones, were able to provide all-in-one-product solutions.

4.1 The "commons" and the standardization process

Out of this came new needs and concerns:

- The need for "commons": these are the common components that reused again and again to avoid reinventing the wheel for each project.
- The need for interoperability. The various parts of the project need to be interoperable, and more importantly, interchangeable. This because customers want to avoid being "locked-in" with a given vendor.

The need for commons led some low-level open source products to appear: these aren't finished products that can be used out of the box, but building blocks that allow to develop faster.

These commons have no strategic value in terms of trade secrets or intellectual property for their users. For example, an XML parser, a web server or even a workflow engine are just "enablers" that allow to implement the actual business logic.

So some people started to work on these commons within open source projects, to share development resources on these basic building blocks.

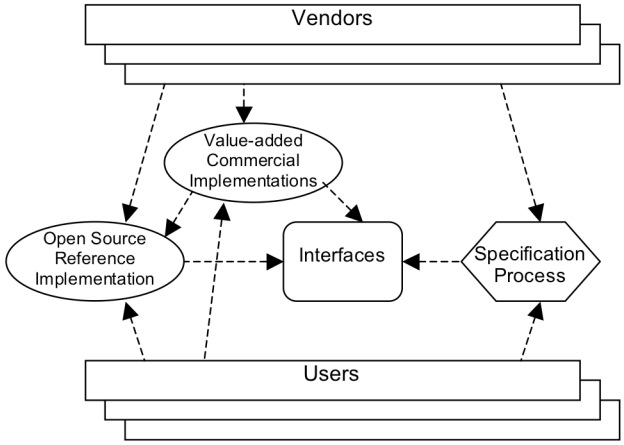
The need for interoperability led software vendors to work together to define standard interfaces between software layers, so that products from various origins could easily be assembled together.

Some standardizing organisations have put open source at the very centre of their process. For example, the World Wide Web Consortium (W3C), that defines the web standards, only makes a specification final once at least two implementations exist for it, so that the ability to actually implement it is proven, and requires one of these implementations to be open source. This ensures that the specification won't stay in the hands of a single vendor, and will be available for free to everybody to evaluate it.

The same applies to the Java platform: the Java Community Process (JCP), that gathers companies and users involved in Java-related products, has to provide a reference implementation of each new specification. Most of these reference implementations are developed as open source products. A key point also is that users are involved in the standardization process along with vendors, thus ensuring that the specification meets their actual needs.

So more and more in the IS domain, there is an offering of open source products that allow new specifications to quickly spread in the industry.

As we'll see later in this paper, this doesn't prevent commercial business and even encourages it. The open source product becomes a commodity available for all, and vendors can provide value-added complements to this commodity.



4.2 Choosing an open source product

Choosing an open source product must be done with care, especially if this product is to have a central role in a company's information system.

The evaluation criteria are both technical and non-technical:

- Consistency with the existing environment: the product must use the technologies that have been chosen or already exist, and should be easily connected, if needed, to the existing environment.
- Maturity: how old is the product? How many stable releases has there been already?
- Project health: what is the release frequency? How many people are in the development team?
- Community health: what is the traffic on the user and developer forums? How fast are user questions answered?
- Commercial support: are there some companies that employ developers or sponsor the project? Do they provide support, services, training or additional products?

These criteria aren't always easy to evaluate, and there's no central directory where to find this information. This likely to change soon with the recent "Business Readiness Initiative" [6] which aims at defining a standard model for rating open source software.

5. Why software publishers move to open source

5.1 Commoditization and complements

When a specification exists for some software layer, a lot of people rely on it, and can therefore join forces to build a common open source implementation of that layer. This implementation then becomes commodity and allows companies to build products of higher value and technological interest on top of it. That's why a lot of software publishers have based their offerings on open source foundations, and participate to the development of these foundations rather than maintaining their own parallel branches. And their participation strengthens the open source group in a virtuous cycle.

A great example of this is IBM, whose Websphere product uses a lot of open source products from the Apache Software Foundation, while still providing some distinctive features that justify its price. We also see some vendors that invented a "non-standard" technology deciding to open source it, or at least its foundations. A good example is BEA with the Beehive project [7], which they donated to Apache. This allows them to avoid vendor-lock in fears expressed by their customers, while still allowing a sustainable business of tools, extensions and services.

Joel Spolsky expands [8] on the effects of commoditization on business strategies: demand for a product increases when the prices of its complements decreases. By collaborating to open source projects, publishers actually reduce the price of their product's complement to zero.

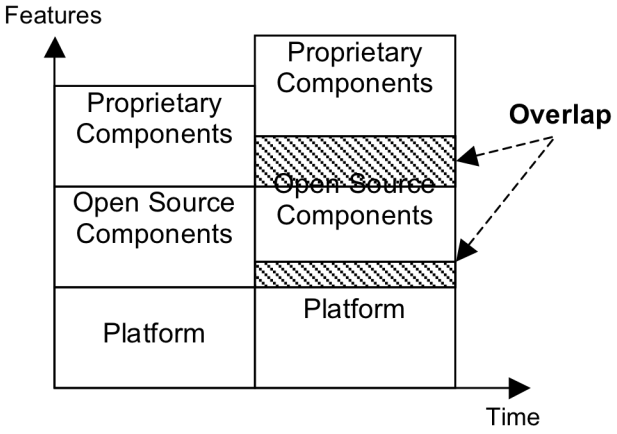
5.2 Dealing with obsolescence

Another phenomenon that drives software publishers towards open source is the technological obsolescence.

A software product is composed of several layers. The bottom layer is the platform: the operating system, the programming language and its standard libraries. Then comes a set of reused components that provide additional features to the platform.

The actual product is built on top of these lower-level blocks, and each layer clearly brings different features.

Now as time goes by, each of these layers evolves and grows, which leads to some functional overlap. These overlapping parts become obsolete, as the lower-level blocks provide the same to a wider range of users and at a lower cost. So how should this overlap be dealt with?



The platform/OSS components overlap is easy to manage: the OSS components that have been obsoleted disappear after a transition period, since there is no more community interest for them. It is to be noted that this overlap is sometime caused by the platform integrating some open source components, such as what happens in Linux distributions or the Java platform.

The OSS/proprietary components overlap is less easy to manage, as it affects the intellectual property and commercial assets of the company. And this is actually one of the driving forces that lead publishers to contribute to the open source software movement.

There are 3 types of answers to this overlap:

- Ignore it. This is probably the worst choice, as the overlap will continue to grow, and at some point in time, the open source solution will destroy the market for the whole commercial product.
- Replace the overlapped part by its open source equivalent. This solution is to be adopted when the open source solution is already mature and well established.
- Embrace the open source solution by contributing to it.

In the IS domain, more and more publishers are going for this 3rd answer. Although it can at first look like giving away some resources and intellectual property, this is actually creating some beneficial conditions for the proprietary product that then becomes a complement to some commodity.

Also, it gives the company some influence in the evolution of the open source components, so that

they can better fit the needs for the upper level proprietary components.

Finally, and this is not to be neglected, it also gives the company an enhanced visibility, and some additional channels to market its offering.

5.3 Developing commons

Embracing open source foundations for a commercial product is also a way to develop commons that are shared between various companies working or even competing in the same domain.

The best example of this is the Eclipse Foundation [9]: in 2001, IBM decided to open source a large part of their commercial IDE, Websphere Studio Application Developer and created the Eclipse project. The reasoning being this move was that modern IDEs were more about providing very sophisticated tools such as project management, requirement traceability, modelling and simulation tools, etc, rather than about managing files in a project and launching a compilation process.

In a few years, an impressive list of companies has joined the Eclipse Foundation (which is now an independent non-profit organization). Many of them are editors of commercial IDEs that work together on a common platform, each keeping its own added value and commercial products on top of this common platform.

This has been explained very well by Patrick Kerpan from Borland in "the IDE is dead, long live the IDE" [10]: for more than 20 years, Borland has been writing IDEs, and each of them needed file management, standard menus such as file/open, file/save, etc. And each new generation of the IDE requires rewriting these low-level features. So when seeing the success of the Eclipse platform, Borland made the strategic decision to collaborate with its competitors on a common platform. This allows them to concentrate on the higher-level distinguishing features that are their real expertise and that customers are interested in.

6. Open source trends in the embedded software world

Open source software is now a key element in the information systems world. Now what about the embedded software world? It's coming. It even has been already there for some years with the gcc compiler, which is used in many cross-development tools, and is the basis of the GNAT Ada compiler.

But the embedded software domain is also very different from the information systems, as embedded systems both need to be carefully optimised, and often need to be certified.

That is why, contrarily to the IS domain, open source is coming first through tools rather than runtime components.

6.1 Tools as complements to realtime operating systems

Realtime operating systems vendors mainly sell runtime licenses of their operating systems. Tools are there to help selling runtimes by making development easier, but aren't their primary differentiating feature on the market.

A few years ago, QNX decided to join the Eclipse Foundation and build there the C/C++ development environment. By doing this, just as Borland, they benefited from a huge set of features brought by the platform, and shared resources with other actors interested in seeing support in Eclipse for these programming languages. They of course kept their commercial IDE offering, which is now a thin layer dedicated to their particular operating system.

About one year ago, Windriver, a major player in the realtime operating system arena, also joined the Eclipse Foundation. And we now see QNX and Windriver, competitors, working together on a common platform, just as Borland did in the IS domain.

6.2 Long lasting development environments

Embedded software often has a lifetime far more longer than IS software, and maintenance has to be possible for many years or even decades. So industrials must ensure of the continuous availability of the tools used to build the software, which can't be guaranteed by proprietary tools sold by often-small companies.

To solve this problem, we see the emergence of projects such as Topcased [11], led by Airbus and gathering companies working on a common and open source toolset for the development of avionics software.

Topcased is a generic modelling toolset based on the Eclipse platform. It is actually a meta-modeller as it allows to define the "model of a model" to quickly build a graphical editor for a particular model. The goal is to provide editors for all models that participate in the design of an aircraft's software. Without the open source commons provided by Eclipse, this project would never have been started.

When presenting the project, people from Airbus are often asked: "since it is open source, what if Boeing decides to use Topcased too?" And the answer is

similar to the one from Borland in the previous chapter: the competition is not on the modelling tools, but on what is done with them. Also, only standard models are open source. Airbus-specific ones are kept proprietary.

Again we see the value of open source commons that help go further in the upper levels where the actual commercial competition is.

6.2 Embedded runtimes: the certification issue

Compared to other domains, embedded software systems must be carefully certified before being deployed, both because such software often has some critical mission regarding human lives and because it is not easily upgraded once distributed in the wild at a large scale.

Certification is a very costly process that somehow goes against the lightweight development model used by open source organizations.

We see however some newcomers in the realtime operating system landscape, that provide certified versions of Linux for the embedded world. So although the software isn't available for free, its cost is dramatically reduced by its open source origins. This cost covers not only the certification process, but also the liability that the vendor endorses.

6.3 Standardizing interfaces

The complexity of embedded software is constantly growing. Today's cars contain dozens of calculators, smart DSL modems containing multimedia applications are spreading in many houses, and mobile phones are as powerful and featured as 10-year old personal computers.

The result of this growing complexity is that not a single company is able to provide the entire software of a system, and also that final product manufacturers no more want or can rely on a single solution vendor.

That's what triggers standardization processes. Two of them are worth mentioning: OSGi [12] and AUTOSAR [13].

Started in 1999, OSGi is a specification for a runtime platform based on the Java technology that can host software services. It defines software contracts that allow application and services developed independently to coexist and cooperate on a single device. OSGi is a consortium of companies from the automotive and set-top-boxes domains, and only members of the consortium were authorized to produce a certified implementation of the specification.

Then, some open source implementations of OSGi emerged. They could not claim strict compliance with the specification, but the fact was they were actually full featured and robust. And this led more people to using OSGi, for use cases that where not initially foreseen. The effect is that although the open source offering competes with commercial implementations, it actually widens the OSGi market. The OSGi group has taken this trend into account, and the latest release of the specification is available under an open source licence, allowing anybody to implement it.

AUTOSAR is a recent initiative in the automotive world that aims at defining an open standard architecture for electronic and software modules. Its purpose is to allow car manufacturers to easily integrate parts from various vendors, to answer the problem of the growing complexity of automotive systems and allow interchangeability of equipments.

The automotive world is not as open as the information systems world, and full open source solutions are not likely to happen in the near future in mission-critical software. However, it is very likely that some strong influence from the open source world will emerge.

A hardware device manufacturer my decide to "share the source" of the associated software with its users and integrators, to allow for faster roundtrips in the software design and be more reactive at integrating new needs coming from users.

Finally, there are a lot of software modules in a car that aren't critical. These are for example all the onboard entertainment features. In these areas, standardized interfaces such as OSGi and AUTOSAR will allow the emergence of open source applications such as multimedia players, enhanced GPS navigation software integrated with the driver's address book, mail readers, etc.

7. Conclusion

In a few years time, open source software has changed the industry landscape in the information systems world. Open source projects are now studied by researchers and management schools [14] to understand how these self-organizing projects achieve their goals and what lessons can be learned from this.

The conclusion that can be drawn from the experience of the information systems world and the current trends in the embedded systems world is that open source cannot be ignored.

Now is it a risk or an opportunity?

For industrials, it is clearly an opportunity, as it allows the development of shared solid foundations and fulfils the need for long-term maintainability.

For vendors, it is a risk if ignored, as it will likely cut down sales. But it is an opportunity for those vendors that will use open source foundations to provide value-added offerings on top of them. But they also have to be aware that business will have to include more service and less runtime licenses.

This will change the embedded software industry landscape. And it already has begun.

8. References

[1] Bruce Perens et al, *"The Open Source Definition"*

http://www.opensource.org/docs/definition.php

[2] The Open Source Initiative: *"Approved open source licences"*

http://www.opensource.org/licenses/

[3] The Apache Software Foundation: "How the ASF works"

http://www.apache.org/foundation/how-it-works.html

[4] Tim O'Reilly: "The architecture of participation", April 2003

http://www.oreillynet.com/pub/wlg/3017

[5] The Apache Software Foundation Incubator

http://incubator.apache.org/

[6] BRR – The Open Readiness Rating for Open Source

http://www.openbrr.org/

[7] Cliff Schmidt: *"Beehive, A case study for new open source business models"*, ObjectWebCon, Lyon (France), 2005

https://wiki.objectweb.org/ObjectWebCon05/attach?page=Sessions%2Fbeehive.pdf

[8] Joel Spolsky, *"Strategy Letter V"*, June 2002

http://www.joelonsoftware.com/articles/StrategyLetterV.html

[9] The Eclipse Foundation

http://www.eclipse.org/

[10] Patrick Kerpan, David Intersimone: *"The IDE is dead, long live the IDE"*, EclipseCon, Burlingame (USA), 2005

http://www.eclipsecon.org/2005/presentations/EclipseCon2005_7.3.pdf

[11] Topcased project

http://www.topcased.org/

[12] OSGi – Open Systems Gateway Initiative

http://www.osgi.org/

[13] AUTOSAR – AUTomotive Open System ARchitecture

http://www.autosar.org/

[14] Free/Open Source research community

http://opensource.mit.edu/

9. Glossary

ASF	Apache Software Foundation
FLOSS	Free/Libre and Open Source Software
IS	Information Systems
IDE	Integrated Development Environment
GNAT	GNU Ada Translator
GNU	GNU's Not Unix (recursive definition)
GPL	General Public Licence
OSS	Open Source Software