Open standards, open source, open minds, open opportunities

Bob Sutor is the Vice President of Standards and Open Source for the IBM Corporation. The postings on this site are his own and don't necessarily represent IBM positions, strategies or opinions, especially if they are about the guitar, fishing, house painting, and musical tastes.

2006 Feb 15 12:30 AM

Regarding the "Open Standards vs. Open Source" series

posted by Bob Sutor

I'm in San Francisco, having arrived about three hours ago for the OSBC conference. On the plane out here from New York I pretty much finished Part 2 in my series on "Open Standards vs. Open Source." It's been several weeks since Part 1 appeared, so I wanted to make sure people knew it wasn't an orphan. I toyed with several ways of presenting the material and ultimately decided to do it in four sections. The second part that I just put up looks at software, particularly traditional commercial software, and talks about the revenue generating ecosystem. By explaining the various bits of the business, I'm going to be be better able to contrast what happens in the open source model of software development and distribution. That's in Part 3. The final part will bring Services Oriented Architecture, or SOA, into the picture and talk about how that is driving the movement toward openness. With luck, the subsequent parts will have less time between them than the first two did.

This series is meant to be non-technical but still, I hope, be relevant and informative for people who don't live and breathe open standards and open source all day long.

Part 1:

2006 Jan 25 11:41 PM

Open Standards vs. Open Source, Part 1: Standards

posted by Bob Sutor

- Part 1: Standards
- Part 2: Software
- Part 3: Open Source Software

- Part 4: The SOA Connection

In my wish list for 2006 that I blogged about a couple of weeks ago, the very first item was my desire for people to have a better understanding of the difference between open standards and open source. Authors have written entire books about these topics, and more are being written even as I pen this. We saw in some of the public hearings in Massachusetts some confusion between what a standard does and what the rules are when you go to implement it. I've been told secondhand that some people were even told things like "if you implement that standard, then all your software has to be given away for free." This, of course, was wrong and confused, if it wasn't, in fact, outright FUD (creation of Fear, Uncertainty, and Doubt). So let's talk about some of the basic notions about and the differences between standards and open source. In this part, I'll focus on the former and start the discussion of software and open source in Part 2.

As I said in the wish list, a *standard* is like a blueprint. It provides guidance to someone when he or she actually builds something. Standards are not limited to software, but are an important part of computer hardware, telecommunications, health care, automobiles, aerospace, and many areas of manufacturing. One of the reasons an architect produces a blueprint is that a builder, an engineer, or an inspector can look at it and say "if you build according to this plan, it will be safe and the house won't fall down." In the same way, some standards are for safety, especially where they involve electrical or electronic components.

A standard is more than just a blueprint, though, because it has to be something with which a lot of people agree. Something that may not have this sort of "blessing," or common agreement, is usually called a *specification.* By abuse of language, we will sometimes call a standard a specification. Put another way, all standards are specifications, but not all specifications are standards.

Standards are also employed when we have to ensure that things made by different people will either work together or work in the same way. I live in the United States, and when I go to the store and buy a telephone, I know that the telephone wire will plug into the jack in my wall. I don't need different jacks for phones made by different vendors. The design of the telephone jack and the plug are not control points for any phone vendor. I make my choice based on the features of the phone, the color, the price, whether it is wireless or not, and so on. There are standards that describe the "blueprints" for the plugs and jacks, but the standards themselves are not the actual plugs or jacks. We separate the ideas of "a standard which may be implemented" and "something that is an implementation of a standard."

To bring this back to software, the OpenDocument Format is a standard, a blueprint, created by a technical committee at the OASIS consortium. It is not software, but rather a description of how you should write out the information in word processing documents, spreadsheets, and presentations should you ever need to put them on disk or, say, attach them to emails. The same description also tells you that if someone gives you a document, spreadsheet, or a presentation and they tell you it is in OpenDocument Format (perhaps via the file extension), then your software applications know what to expect when they read the file. Because it is a standard, the information can be used by anyone who builds software that complies with the standard. No one vendor can arbitrarily change it, and that provides a lot of security for people who save their documents in OpenDocument Format.

Another important standard is HTML, the rules by which you format pages for the World Wide Web. Although there were vendor differences in HTML in the mid 90s, most people no longer think that it is reasonable to allow vendors to break interoperability by implementing too little of the standard or doing their own special things. We don't need different browsers to view web pages from different people, though some browsers like Firefox and Opera are known to adhere to the web standards, essentially the blueprints for the web, better than other browsers. To extend my earlier analogy, we know that a web page ("the plug") will fit into the browser ("the jack") and then I can see and interact with the page ("I can talk on the phone").

Just as we said in the 1990s that no vendor should have a control point by having their own flavor of HTML, we are saying the same thing about document formats in the 2000s. In fact, since word processors have been with us a lot longer than the web, it's surprisingly that standards weren't created, if not demanded, there earlier.

So, to summarize, a standard is a blueprint or a set of plans that can be implemented. Where do standards come from?

A *de facto standard* is a specification that became popular because everyone just happened to use it, possibly because it was implemented in a product that had significant market acceptance. The details of this specification may or may not be available publicly without some sort of special legal arrangement.

The basic problem with a *de facto* standard is that it is controlled by a single vendor who can, and often does, change it whenever the vendor decides to do so. This frequently happens when a product goes from one major version to another. At that point, everyone else who is trying to interoperate with the information created in the owning vendor's product must scramble and try to make their own software work again. This is easier, of course, if they can actually see the new specification and there are no impediments, legal or otherwise, to implementing it. The owning vendor gets a time-to-market advantage, possibly increasing its marketshare, again.

Traditionally, it was not in the interest of the owner of a *de facto* standard to make the details too widely available because they didn't want to make it easier for anyone else to move into their market space. They would say, "Why would I voluntarily let other people build products compatible with my data? They might steal away my customers!" To turn this around, it is not in the best interests of customers to be locked into *de facto* standards controlled by a single vendor. The customer might say, "I may have used your software, but it is my information, and I very much want and demand the freedom to use any application I want to process my information." *De facto* standards decrease customer empowerment and choice, though they linger on.

The second kind of standard I'll mention is something I'll call a *community standard.* As you might guess, this is something created and maintained by more than one person or company. The members of the community may work for companies or governments, belong to organizations, or may be experts who are otherwise unaffiliated. The standards creation process involves negotiation, compromises, and agreement based on what it best for the community and the potential users. It is a classic fallacy to think that this necessarily creates a “lowest common denominator.” Smart people can make good decisions together, even if they don’t all work for the same company. Conversely, people who all work for the same company don’t necessarily always make smart decisions. They might, for example, produce *de facto* standards that have security vulnerabilities.

Community standards usually get blessed, as I termed it above, by being created or submitted to a *Standards Development Organization,* or *SDO.* While it does happen that people may get together and write a standard from scratch in an SDO, it is very likely that one or more parties will bring drafts to the table as a starting point. It is usually expected that the developing standard will change over time as more minds are directed at the problem that the standard is expected to help solve. Standards go through multiple versions: it is not uncommon for the first version to take one to two years and then about the same time for each of the next one or two iterations. At some point it will stabilize and either become fairly universally used or else become eclipsed by an alternative way of tackling the same general problem. For example, the new web services standards are starting to be used for distributed computing, replacing older standards, as Service Oriented Architecture becomes more broadly deployed.

I want to return to this “community” idea for a moment. If you bring something to an SDO, you take a risk that others may change the specification, perhaps in ways that interfere with your product plans. One word: tough. Working within a community does not mean walking in and saying “I’m the king (or queen) and you can’t change anything unless I say it is ok.” The value of creating a standard in a community is that products from different sources can work together to build solutions that solve real customer problems. If you can’t compete by creating superior, higher performing, more scalable, more secure products and perhaps the services to give the customers what they need, then I would suggest you have problems beyond not controlling the creation of a standard. What you are basically saying is “I can’t win on a level playing field.” Your customers might be interested in hearing that.

Some SDOs are *de jure* organizations: they have particular credentials in national or international settings. Some governments have laws that make it very difficult to use standards that do not come from *de jure* organizations. ANSI, ITU, and ISO are examples of *de jure* organizations while groups like the W3C, OASIS, and the OMG are usually just referred to as consortia. Sometimes a standard produced by a consortium will be submitted and blessed by a *de jure* organization to make it more palatable for government procurements. Of course, *de jure* organizations, like all standards groups, must be very careful what they bless because they have reputations for quality and relevance that they hope to maintain.

The last topic on standards that I want to address is what it means for a standard to be *open.* I’ve spoken about this previously in the blog but, at the risk of repeating myself, let me say that I think we need to consider five aspects of standards and ask some important questions about each of them:

- How is that standard created?
- How is it maintained after Version 1.0?
- What is the cost of getting a copy of the standard?

- Are there restrictions on how I can implement the standard?
- Can I use just a part of the standard or extend it and still claim compliance?

In answering these, we need to think in terms of transparency, community, democracy, costs, freedoms and permissions, and restrictions.

- The more transparent the standards process is, the more open the standard is.
- The more the community can be involved and then actually is involved, the more open the standard is.
- The more democratic the standards process is, where the community can make significant changes even before Version 1.0, the more open the standard is.
- The lower the standards-related cost to software developers who want to use the standard, the more open it is.
- The lower the standards-related cost to the eventual consumer of software that happen to use the standard, the more open it is.
- When the licensing of the standard is more generous in the freedoms and permissions it provides, the more open the standard is.
- When the licensing of the standard is more onerous in the restrictions it imposes, the less open the standard is.

From these and perhaps other criteria, we should be able to come up with some sort of "Standards Openness Index." In the meanwhile, use them when deciding for yourself how open a particular standard is.

At this point I hope you have a good sense of what an open standard is. I also hope you understand there is a spectrum of "goodness" for several aspects of how a standard comes into being and how it might get used. "Open" has become a standard marketing term, so make sure you ask good questions of those who are trying to convince you that they are as or more open than the other guy.

In Parts 2 and 3, I'll talk about software development and we'll start to ask questions about openness in terms of software implementations. Just as we saw with standards, there is a spectrum of "openness" when it comes to the creation and use of software. Open source software is a large category, but I'll focus on some of the basic issues and how open source can be related to open standards.

One last thing to think about: near the very top of this discussion I talked about how a standard can be considered a blueprint. Do people ever freely give away things they make from blueprints? Does that make business sense? Did you ever sign up for a cellular phone service and get a free phone? Why didn't they charge you? Do they make their money elsewhere?

Part 2:

Feb 15 12:07 AM

In Part 1 of this discussion of open standards and open source, I focused on what a standard is and what it means for it to be open. I compared a standard to a blueprint and said that we separate the ideas of "a standard which may be implemented" and "something that is an implementation of a standard." Open source is a particular way of implementing and distributing software, enabled by legal language that gives a range of permissions for what people may do with it. Just as we saw in the first part that there are a number of factors that can determine the relative openness of a standard, so too are there conditions that allow, encourage, or even demand behaviors from people who use or further develop open source software. In this part I'm going to look at traditional commercial software and then in Part 3 contrast that with open source.

I'm going to assume you have an intuitive sense of what software is, though we'll return to that in a few moments since it is important for understanding what one does with open source software. Software is what makes the physical hardware in a computer do its magic. That is, software might be a web browser, an online customer sales tool, a video game, a database that supports your local bank, the reservation system for an airline, the invisible commands that respond to your actions in your handheld MP3 player, or even what controls how your automobile changes its mix of gasoline and air as you start to drive up a hill. These are just examples, and I'm sure you can think of many more.

You can also probably think of several different "titles" of software that are in the same category, such as *Halo, Doom,* and *Syberia* in the video game category, and *Microsoft Word, Corel WordPerfect,* and *Lotus WordPro* in the word processor category. These are all examples of commercial software: you pay money and you get a legal license to play the game or write a novel by using the software. The license gives you certain rights but also places certain restrictions on what you can do. The features and value of the software, the cost, and the rights you obtain determine if you pay for the license. Similarly, if you don't find the restrictions too onerous or unreasonable, then you might be inclined to agree to get and use the software.

Depending on how you look at it, some of the rights might be seen as restrictions. For example, if the license says "you have the right to use this software on one and only one computer," then you are restricted to not putting the software on two or more computers. From the perspective of the company that wrote and licensed you the software, this is probably very reasonable because the company (the "developer") wants to be remunerated for its efforts and its costs for creating the software. The solution to running the software on multiple computers is easy: you make some sort of deal by paying more. Minimally, if you paid $50 for one copy then you might expect to pay $100 for two copies, $150 for three, and so on. You might be able to do better than that, particularly if you are dealing directly with the software provider and you need a lot of copies. Tell them you need 1000 copies and see what they offer you!

Ultimately, the provider of the software wants to get paid and even make a profit, and probably the more the better as far as he or she is concerned. It can be complicated to figure out how much to charge, but sometimes the market will decide for you. For example, my son William has a handheld video game machine and games for that never cost more than $40. A vendor could try to charge more for a game he or she creates for the machine, but if potential customers think it is too much, then they won't buy it and the vendor won't recoup his or her development costs. If the vendor charges too little and the game becomes very popular then he or she might be "leaving money on the table." That is, the vendor might have been able to generate greater revenue and profit by charging as much as the market would bear.

Sometimes charging less for a video game can cause more people to buy it. If there is an aftermarket for things like game hint books, stuffed animals of the game characters, and a movie deal, then the software provider might significantly lower the game price to get as many people as possible playing the game. The real profit might come from the aftermarket products. Might the price ever go to zero? In theory yes, but it rarely happens when the game is new. Nevertheless, it is an interesting idea to give a game away if you can generate a tremendous amount of money in other ways.

This sort of thing is not entirely unheard of: my family has bought boxes of cereal that had free DVDs in them and the local fast food restaurant has given away DVDs with certain meals for children. In this last example, if the DVD promotion is successful enough that it significantly drives up the sales of food, then the restaurant chain might end up paying quite a bit of money to the distributor of the film or films.

Business models around software therefore are more sophisticated then simply saying "I will get all my revenue from what I charge to license my software." The software developer is interested in the total revenue, remember, even if not all of it or even the majority of it comes from licensing.

Thinking beyond the video game category, we can see other ways to make money in the software business. One is support. What happens when the user of the software has a problem? He or she might search the Web for an answer or, failing to find a solution, might call the software provider for assistance. Is this assistance free? It depends. Sometimes there is free support for the first thirty or more days after you bought and registered the software. Another option is to give the user one or two free calls and then charge after that on a per phone call/time taken to resolve the issue basis. Yet another option is to charge a fixed fee for a year's worth of support, though there may be limitation on how many times you can call.

It's not uncommon to have tiers of support levels, from something fairly minimal to something which is comprehensive. The more you get or think you are going to need, the more you pay. There can be significant money in providing support. In fact, if the support just concerns how to use the software, there can be companies with no relationship to the original software provider that provide support packages. In this way, a particular piece of software starts to create what might be termed an "industry" that extends beyond the original developer.

What happens if there is fundamentally something wrong with the way the software operates, a "bug"? Someone with access to the original source code must go in and figure out what the problem is, find a solution, and then distribute either a new version of the software or something that fixes the specific area that is causing the problem. If you do not have the source code from which the software was created you can: 1) wait until the original vendor decides to fix it, which may very well be the best solution for non-critical items, 2) find a work-around, that is, another way of doing what you wanted, or 3) switch to an entirely different application that does not have the problem. There can be many kinds of problems, but security and data corruption ones are especially serious.

If you had access to the source code for the software, could you fix it yourself? Maybe, or you might be able to find or pay someone to do it for you. Are you concerned that the provider of your software might not be in business forever and so you want the extra insurance of having the source code in case you need it eventually? Maybe. You might sleep better at night, as they say, by having the source code but you also might sleep just as well if you had a vendor you trusted and knew was going to be around for a long time. You will need to decide for yourself. You should make this decision from practical business considerations, not political or ideological reasons. We'll return to this in Part 3.

Another area where a software provider might be able to generate revenue is in services. If I decide to buy new accounting software, how do I get the information from my old system to the new one? How do I connect the new software to the customer relationship management (CRM) software that my sales team uses? These tasks might be easy to accomplish but, then again, they might not be. You might have the expertise yourself or on your staff to do this work but, then again, you might not. To whom do you turn for help in making your systems work together? It's possible that the provider of your new software will do the upgrade and integration work for you, but they may not do it for free. There may be other service providers who can do the job better or at a better price. Perhaps there was a company that put together the various hardware, software, and networking components on which you run your organization. It might be a good idea to pay them to do the transition to the new software. Again, there may be someone who provides value to you around the software who is not the original developer. The more successful and widely used the software is in the market, the more service providers there will be. This can be a bit of a "chicken and egg" problem because the software might not become successful until there are enough service providers to help customers get the most value from it. This is one reason why software vendors like to have many business partners.

The partners may provide software rather than services. It's not uncommon for a partner to specialize software for use in a particular industry. For example, the partner may work in the legal industry and provide special templates that make it easier to produce a broad range of legal documents with consistent formatting. They might provide document management software so that the attorneys can find and retrieve documents. In this case again, the original word processing software enabled others to develop an aftermarket. The aftermarket can further increase sales of the software, setting up a "virtuous cycle" in which everyone makes money and the software might achieve commanding market share.

I went through this to show you how the creation, distribution, and use of software can create economic value and revenue that extends beyond the initial price that is charged for a license. Anyone who only looks at the license cost is not thinking broadly enough about the big picture.

Software quality is also important. If a customer support application crashes often and loses important data, then it probably isn't too important that it is less expensive than a more stable product. Conversely, higher quality software that costs less than the competition deserves to gain significant market share. In particular, software with security and privacy problems can sometimes not be worth any price.

Software features can affect price, though it is not the total number of features that is important. Rather, are the features that people really need well implemented? Are they easy to use? If I only use 20% of the features of a piece of software and I can find another application that only implements the features I need and costs less money, should I go with that?

Before I leave this discussion about commercial software, I want to talk again about standards. Standards make things work together. In an example above, I talked about integrating a new accounting system with my sales' team CRM software. How exactly do you that? If both pieces of software come from the same vendor, it might have some special language and protocols that link the two together. The vendor will probably claim that this makes everything work more efficiently. This is dangerous.

Proprietary schemes for connecting software make it very difficult to substitute software made one vendor with software made by another. This is exactly what the vendor that creates the accounting and the CRM software wants. This gives control of your system to the vendor and limits your choices. Again, this is what the vendor in question wants. This situation is called "lock-in" and, as I said, is dangerous. You want the ability to choose the best software for your organization, no matter what the cost or who the provider is. These proprietary schemes can become the *de facto* specifications I discussed in Part 1.

When the industry agrees to common ways of having software from different providers interact, then you the user, the customer, the consumer, the citizen wins. To repeat myself for emphasis, open standards allow software made by different people to work together and gives you choice and control.

One of the ways in which software works together is by sharing information. It should be a hallmark of modern software that the way this information is represented should not be proprietary. You want the ability to use any software that you wish to interact with your own information. The price of the software is irrelevant. More expensive software should not be accorded more leeway in using proprietary ways to represent your information. Similarly, software with greater market share should not force more restrictions on who and what can use your data.

Through this discussion I've highlighted several important aspects of software, and have focused on examples of software for which you pay for a license:

1. The license fee can be larger or smaller.
2. The license fee can affect market share.
3. Market share might be increased by a lower license fee.
4. Support is one way of generating revenue from software, and it is not limited to the original developers.
5. Service delivery is another way of making money in the software business and neither is it limited to the original developers.
6. Open standards are important for allowing software made by different people to work together.
7. The price of software is irrelevant when thinking about whether open standards should be used: they should. End of story.

What would happen if the price of the software was zero (in any currency)? What if you had access to the source code and could make any changes you wished? What if you could build new software on top of software created by others without paying them? Is this anarchy? Will capitalism survive? How will we make any money? We’ll look at these issues in Part 3.