NICHOLAS M. STOUGHTON

why standardize?

USENIX Standards Liaison

nick@usenix.org

Descriptive or prescriptive? Should a formal International Standard describe a single existing product, or should it prescribe how future products should work? Is it an abuse of the process for any one company to force a document describing its product through the standards mill?

Most of the formal standards on which I work describe some sort of an interface. They act as contracts among multiple producers of the interface and multiple consumers of its services.

For example, POSIX describes the interfaces provided by a wide variety of different operating system implementations (e.g., Solaris, AIX, HP-UX) and portable applications that should compile and run on any of those implementations. The C standard describes the language a compiler must be able to translate and provides a guarantee that an application that strictly conforms to the language standard will always run the same when translated with a strictly conforming compiler.

As such, standards such as POSIX or C are *prescriptive*; they dictate behavior on both sides of the interface. They tell an implementer what must be done in order to conform, and they tell an application developer how to write portable code that does what was intended.

Occasionally, however, a *descriptive* standard is needed. The Linux Standard Base (LSB) is one such standard. It describes the binary interface between an application and a Linux distribution. It is still prescriptive in the sense that a distribution that fails to ship the versions of the libraries or symbols required does not conform, but it does not tell implementers how to write interfaces. Instead, it describes what the implementation does. Linux hopes to be POSIX conforming (and almost is), so POSIX can tell implementers how to write interfaces; the LSB only describes the size and shape of the binary interface itself.

When we are developing the POSIX standard, however, every attempt is made to take into account existing and historic practice (including Linux behavior). In that sense, even POSIX ends up being somewhat descriptive. POSIX tries to specify the things that every implementation must do (by looking at what every implementation actually *does*), and it leaves unspecified those areas where implementations may differ.

But when a standard is descriptive you have to start asking yourself, "Who is the intended user?" and "What is the intended purpose?" If the standard simply describes something that *is*, a single product with a single implementation, rather than the way a conforming implementation should work, why bother? The whole purpose of the standard is to allow competing implementations and to allow portable applications. In the LSB case, the answer to the question is easy; the intended user is any application developer targeting the Linux platform. The purpose is to ensure binary compatibility among distributions. It does prescribe what distribution vendors must implement in their products, and it does promise conforming application developers

binary portability across conforming distributions.

But let's look for a few minutes at another standard wending its way through the process at present: Office Open XML (OOXML). The name is (deliberately?) confusing: It is Office Open, and *not* Open Office! This is a standard developed by Microsoft for their Office product.

At this point, OOXML is an approved standard within Ecma International (the people who brought you Ecmascript). It is sometimes known as Ecma Office Open XML, or EOOXML. The Ecma standard (Ecma-376) is being submitted to ISO/IEC JTC 1 via the "Fast Track" process. If this process were to be completed, Ecma-376 would be an international standard that describes a file format used by Microsoft Office. No other product could ever fully conform to it. There is no good technical purpose for this standard. Of course, the Ecma committee doesn't actually *say* that anywhere, but should you attempt to read the 6,000+ pages, you would find numerous places with phrases such as "This element specifies that applications shall emulate the behavior of a previously existing word processing application" (Microsoft Word 95), but nowhere will you find a description of what that behavior is. If you are trying to write an application that reads an OOXML file, what chance do you have of implementing this? In Microsoft's defense, most of these requirements are marked as "deprecated" and describe "compatibility" settings. However, they are still a mandatory part of the standard for conformance. How else are they going to achieve their goal of having every document ever produced by MS Office capable of being expressed in OOXML?

What the Ecma committee *does* say is:

> The goal of the Technical Committee is to produce a formal standard for office productivity applications within the Ecma International standards process which is fully compatible with the Office Open XML Formats. The aim is to enable the implementation of the Office Open XML Formats by a wide set of tools and platforms in order to foster interoperability across office productivity applications and with line-of-business systems.

Sounds great, right? But it is a lofty goal that they have singularly failed to achieve or even approach. In their press release announcing that the standard had been approved by Ecma International, they say, "The new open standard safeguards the continued use of billions of existing documents." And that's the point: Safeguard MS Office documents. Any legacy MS Office document can be converted, by MS Office, into the OOXML format, and any conforming application must be able to handle the result.

The charter for OOXML expressly locked the standard to Microsoft Office. (Recall that bit about "fully compatible with the Office Open XML Formats.") So the entire standardization process within Ecma International was tied up by the language that expressly forbade any attempt to deviate or improve on the Microsoft format. The process was not open, and it does not represent industry consensus (both stated goals of the ISO/IEC process).

OOXML describes a file format. Granted, a file format standard is somewhat different from a straight interface type of standard. But it's not that different: It should be possible to write an application that produces such a file and know that any other conforming application can read the same file and produce the same results. In the OOXML case, there can only be one application that can produce or consume the file: Microsoft Office. To pretend that this is a standard that will increase document portability is an outright lie.

And to make matters worse, ISO/IEC already has a standard that is in exactly the same space, is well written, and is well adopted: Open Document Format (ODF). As I write this, OOXML is at the start of a six-month ballot. The first thirty days of that period is to note "any perceived contradiction with other JTC 1, ISO or IEC standards." National bodies (e.g., those of the U.S. or the U.K.) can submit comments on such "perceived contradictions," and if any are received, the five-month remainder might be put on hold. Groklaw has been compiling a list of contradictions to submit. We shall see what happens.

There can only be one purpose in Microsoft attempting

to push the OOXML standard through: to be able to state that it has a standard conforming product. It is a simple knee-jerk reaction to ODF, the accepted standard that has locked them out of some major government contracts. And there can only be one user of the standard: Microsoft.

Now, don't get me wrong; I'm glad that Microsoft has published a specification that tells others the nitty-gritty details of one of the file formats used by Office. It just doesn't need to be an international standard. The only possible beneficiary of the effort to make it one can be Microsoft. The company has even managed to get some very clever wording into the legal license arrangements that appears to prevent unauthorized (by Microsoft) implementations.

This controversial standard is not without its supporters, and not entirely without merit, at least at the theoretical level. There's more than one programming language, so why shouldn't there be more than one office document format? And ODF is not perfect in every respect.

The OOXML document really sums up everything that is wrong with the ISO "Fast Track" process, and it may indeed even lead to the downfall of that process. The Fast Track process was originally designed to allow a standard developed through an open process in another standards development organization (SDO) to speed through the ISO process. In theory, the document had already accepted adequate comment and review from that SDO itself.

However, some SDOs (and Ecma is a particularly notable one here) have simply served as backdoor ways to get a poorly reviewed, proprietary document published by ISO. If OOXML gets through these next six months unscathed, the Fast Track process will be seen by many as utterly worthless. The process is used as a sort of shell game: The message to Ecma members is, "Oh, don't worry if this isn't perfect . . . it will be reviewed again by ISO," while saying to ISO members, "Oh, you don't need to put any effort into this document in ISO . . . it has already been reviewed by Ecma."

An interesting perspective (and my motivation for this article) can be found at http://www.robweir.com/blog/2006/01/how-to-hire-guillaume-portes.html.