Lessons from the Adoption of MusicXML as an Interchange Standard

Michael Good
Recordare LLC
PO Box 3459
Los Altos, California 94024 USA

Abstract

Around 2000, many people realized that XML technology could be a great way to finally create a successful interchange format for music notation and digital sheet music applications. In the past, adoption of music notation interchange formats had suffered from both technical and social problems. Previous efforts like SMDL and NIFF never met their goals of becoming a standard music notation interchange format, even with the ISO 10743 seal of approval for SMDL.

The major technical problem was to design a format that was complete enough for both commercial and academic use, while usable enough to be approachable for developers. The major social problem was how to get the format adopted by the market leaders in music notation editing, both of whom were inclined to keep to their proprietary formats. MusicXML overcame these technical and social problems to become the first successful interchange standard designed for music notation applications. As of December 2006, MusicXML works with over 60 music applications, including all the market leaders for music notation editing and scanning. None of the other XML formats for music notation interchange have been adopted by applications outside of their own organizations.

Other application areas have similar adoption challenges, especially where market leaders might view a standard XML format as weakening a proprietary hold on customers. Examining how MusicXML overcame these technical and social barriers to become the de facto interchange standard for digital sheet music applications reveals some lessons that can be applied to other XML-based interchange formats:

- Apply usability techniques to language design in the same way that you would to GUI design. Use terms that domain experts understand. Where possible, avoid introducing concepts that are specific to applications in the domain, rather than to the domain itself.
- Develop the format together with software that reads and writes the format. Use evolutionary delivery to deliver early versions of both the format and the software, with frequent iterations to get both working well together. This particularly helps for meeting the twin goals of completeness and usability.
- Make sure that the format can exchange data in both directions with at least one market leader in your application area. Do this as early as possible in your development process. Do not expect the market leaders to do this for you.
- Market to other developers who would get a business benefit from using your format. This will be difficult until your format can exchange data with a market leader.
- Give format developers good support. Involve the development community in the design of the format. Be responsive to developers who are using early versions of your format and translation software.
- Avoid overhead. In a small market like music notation software, organizations like ISO and OASIS can be too costly in both time and money for even the largest industry players. Following the model of Adobe's PDF format - an open format under single company control - can be more effective in these situations.

An Introduction to Music Application Formats

The iPod phenomenon has brought the use of computers for music applications to more people than ever before. Computers can represent music in two basic ways:

- An *audio* file represents music as sound. This is the type of the representation used in iPods, CDs, and other computer applications that play recorded music.
- A *symbolic* file represents music in terms of musical concepts relevant to performers who read music. This is the type of representation used in music notation applications like Finale and Sibelius, or digital sheet music applications like MusicNotes and musicRAIN.

Converting symbolic files into audio format is a booming business for music notation software developers, especially for classical composers who need to create demos in order to interest performers in their music. Symbolic files can also be converted into image-based formats like PDF and JPEG. These formats can display the printed music notation, but contain no musical semantics.

Converting audio files to symbolic format, on the other hand, has more in common with some of the most difficult problems in artificial intelligence. No applications solve this problem satisfactorily at this time in any but the most restricted circumstances. Symbolic and audio formats are related, but they are not interchangeable.

Many types of music applications use symbolic formats:

- Notation editors like Finale and Sibelius are used both to compose music and to prepare music for performance and publication.
- Music scanners like SharpEye Music Reader, SmartScore, and PhotoScore convert printed music into symbolic format using optical music recognition (OMR), similar to converting printed text into computer text files using optical character recognition (OCR).
- Sequencers like Cubase and Logic combine audio and symbolic formats for composers who work more independently of music notation. MIDI is the symbolic format of choice for these applications. Sequencers focus on sound output whereas notation editors focus on printed output. Notation-based sequencers like Notion are bridging the gap between these two categories.
- Digital music stands like MuseBook Score and OrganMuse display music electronically, listen to the performance, and automatically turn pages as needed.
- Digital sheet music applications like MusicNotes and musicRAIN are used to sell sheet music online. They usually offer the ability to play a piece of music and transpose it into a particular key before printing the file.

In the music market, the Macintosh platform is much more important than in many other application areas. This is especially true for high-end professional users. A successful symbolic music format needs to work with both Macintosh and Windows applications.

For many years, the Standard MIDI File (MIDI, 1996) was the only common interchange format for symbolic music files. However, MIDI was developed in order to exchange musical messages between electronic musical instruments, not to exchange music notation between applications. Therefore MIDI contains sophisticated representations for how to make music sound on an instrument, but primitive representations for how to make music appear on a printed page.

Many people tried to develop music formats to move beyond MIDI to a more powerful interchange format for music notation applications (Selfridge-Field, 1997). The most noteworthy of the 20th century efforts in this area were NIFF (Grande, 1997) and SMDL (Sloan, 1997). Neither of these formats was widely adopted. Music scanners were the only application category to adopt NIFF as a format for saving files. Despite approval as ISO 10743, SMDL was never implemented by a commercial music application.

Symbolic Music Formats Using XML

Around 2000, many people realized that XML technology could be a great way to finally create a successful interchange format for music notation and digital sheet music applications (Castan et al., 2001). The Recordare® MusicXML format was one of numerous proposals. Others included MML (Steyn, 2002), MEI (Roland, 2002), and WEDELMUSIC (Bellini and Nesi, 2001). Like SMDL before them, none of these contemporary XML proposals attracted support from commercial music applications.

In contrast, MusicXML has succeeded in becoming the de facto interchange standard for symbolic music formats. Figures 1 and 2 show the growth in MusicXML adoption between when MusicXML 0.5 was presented at XML 2001 (Good, 2001) and the status as of XML 2006. Even five years ago, MusicXML had achieved more success as a symbolic music interchange format than any prior effort besides MIDI:

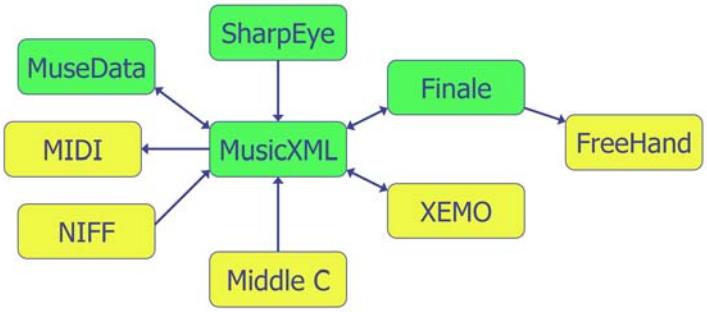

Figure 1: MusicXML Adoption as of XML 2001

However, industry adoption is far more pervasive today:

Figure 2: MusicXML Adoption as of XML 2006

To see the difference that MusicXML makes to musicians, compare how a file transfers from Sibelius to Finale using MusicXML instead of MIDI. Sibelius and Finale are the two dominant professional music notation editors, and it is very common for users to want to exchange files between the two applications. A typical case is when a composer uses Sibelius, but a publisher, arranger, or engraver uses Finale.

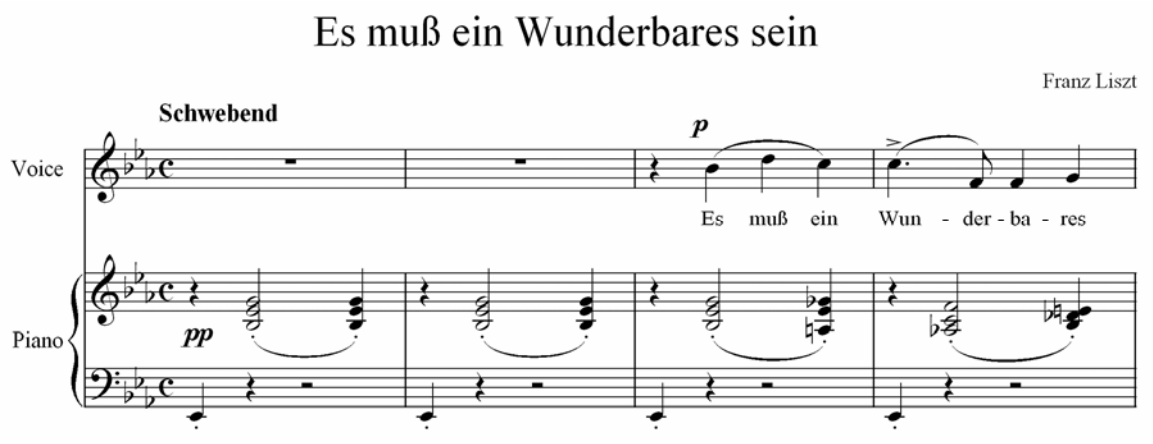

Figure 3: Original Music as Entered in Sibelius 4.1

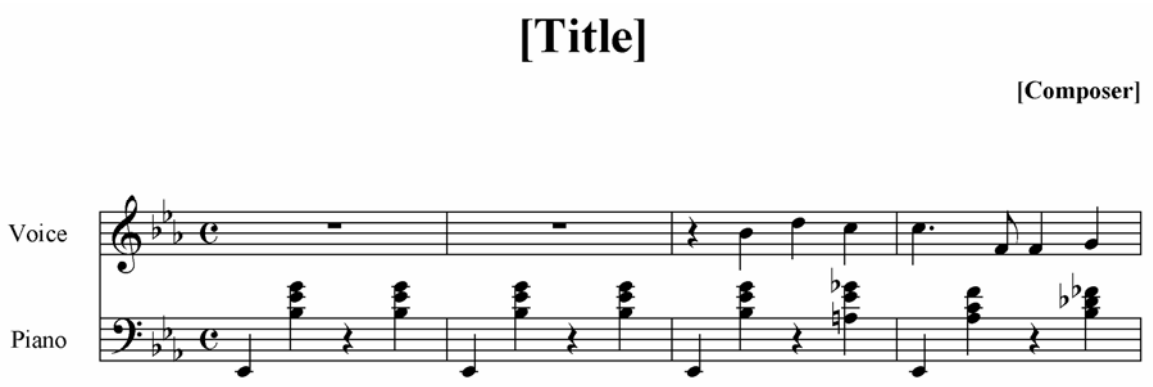

Figure 4: Music as Transferred to Finale 2007 via Standard MIDI File

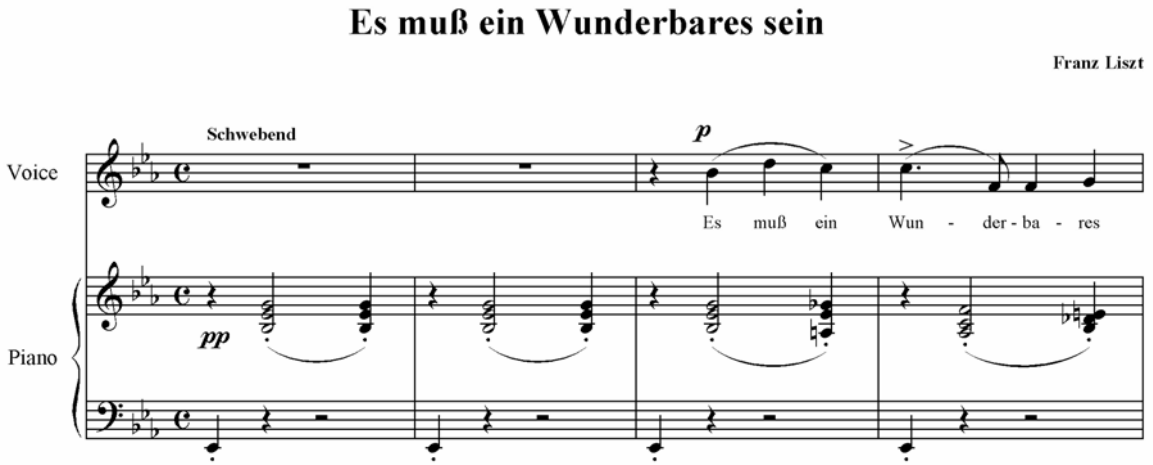

Figure 5: Music as Transferred to Finale 2007 via MusicXML

Figure 3 shows the original Sibelius 4 file. Figure 4 shows the results of a MIDI transfer between Sibelius 4.1 and Finale 2007. Figure 5 shows the results of a MusicXML transfer between Sibelius 4.1 (exported using Recordare's Dolet® 3.2 for Sibelius plug-in) and Finale 2007. The results in Figure 4 are so poor that the Finale user would likely need to start over and re-enter the notes manually. Not all transfers differ this dramatically, but MusicXML transfers between notation applications nearly always save considerable manual rework compared to MIDI transfers. This type of transfer is just one of many current uses of MusicXML within the world of commercial music applications (Good, 2006).

Why did MusicXML succeed when so many prior and contemporary efforts to create a standard interchange format for symbolic music files had failed? The entrenched music application vendors were initially no more receptive to the MusicXML effort than they had been to prior efforts – perhaps even less so, given the previous high-profile failures of both NIFF and SMDL. How was MusicXML able to support both established market leaders like Finale and newcomers like SharpEye Music Reader while still in beta? The answers may prove valuable for designers of potential interchange standards in other application domains, especially those with entrenched market leaders who may not see an interchange standard as being helpful for their business model.

I believe there are six primary lessons to draw from the MusicXML experience, which may be summarized as follows:

- Apply usability techniques to XML language design.
- Develop the format together with the software.
- Support a market leader early.
- Market to other software developers.
- Give format developers good support.
- Avoid overhead.

Apply Usability Techniques to XML Language Design

Usability is an important consideration for designers of XML interchange languages. By definition, interchange languages are going to be used by different communities – communities of users of different applications, and communities of developers of different applications. Usability of an interchange language involves balancing the needs of these different communities.

Usability for application developers is particularly important because no interchange standard can be established without developer support. Application developers are often asked to investigate integrating applications with third-party technologies. An important part of adoption is to make sure that the interchange language feels right to developers during their initial investigation of the language technology.

We believe that several key aspects of MusicXML's design distinguished it from other attempts to standardize music notation formats:

- MusicXML's scope was limited to common Western music notation from the 17th century onwards. This scope is broad enough to cover a wide range of commercially and culturally important music, and matches a limitation on the expertise of performing musicians.
- Wherever possible, elements and attribute names used musical terms, not technical terms specific to computers. Limiting the scope to common Western music notation was a prerequisite for making this possible.
- MusicXML's design prefers clarity over concision. This includes using full names rather than abbreviations, and a preference for structuring musical data in elements rather than attributes.
- The overall structure of MusicXML documents closely matches the structure of many leading symbolic music formats, both in commercial and academic applications. This comes in part from MusicXML modeling a document, not an abstraction of a document.

Limit Scope Carefully

One of the most important design decisions for MusicXML was choosing its scope. The goal was to represent as much music as possible that fell within a common tradition, making it neither too narrow nor too broad. The decision was to represent common Western music notation from the 17th century onwards. This includes present-day popular music notation and tablature, but excludes medieval music and music from other traditions and geographies, such as the many rich musical cultures in Asia and Africa.

Another aspect of the scope decision was the type of applications that would use the format. We wanted MusicXML to be used by any applications that would use a symbolic music format – not just notation editing, but music scanning, sequencing, performing, teaching, and musicological analysis. This had important implications for structure decisions as we will describe below.

Setting scope appropriately is essential to having an interchange language represent documents in a way that is faithful to the documents as understood by the people who created them. Western medieval music, for instance, has very different core concepts than today's Western music. Pitch and rhythm are specified very differently, not only with different symbols, but with symbols that convey different meaning to the musicians that perform from them. For example, the barlines that are central to most Western music are absent from most medieval music, reflecting very different roles for meter. As Dumitrescu (2001) argues, the best approach is to represent medieval music in a separate language that is faithful to the original music, and then use exchange tools to allow conversion back and forth with modernized notation. Dumitrescu's Corpus Mensurablis Musice Electronicum project is an example of such an XML language for representing the mensural notation of European music in the 14th through 16th centuries.

Differences with non-Western music are even greater. Different musical communities have different understanding of pitch and rhythm, and what is "the same" versus what is "different" between two musical performances. Statistical learning forms a large component of how we perceive and respond to music (Huron, 2006). The very different acoustic properties of the world's musics suggest that you do not get universal understanding of music without similarly universal listening.

For symbolic music, the SMDL format is a canonical example of the dangers of setting scope too broadly. Its goals included trying to represent all music – both sound and "notation in any graphical form". But no musician alive understands all music from all time periods. The resulting format is overly abstract, disempowering musicians at the expense of computer theorists. Most symbolic music software application developers are musicians, so making a format that musicians cannot understand alienates both the developer and user communities.

Choose Names Based on the Application Domain

Wherever possible, MusicXML chooses the names for its elements and attributes based on musical terms, not terms specific to computers or to a particular computer application. Of course, the names of terms differ across different languages. In the case of music, the terminology for note durations is very different between the English used in the USA and the UK. MusicXML's names generally use terms as expressed in English from the USA.

However, MusicXML is representing symbolic music for use with computer applications, and sometimes computer terminology is unavoidable. MIDI is particularly pervasive for playback of music in symbolic format. In some cases, MusicXML adopts MIDI terminology directly. More often, though, the musical concept is defined a bit more broadly, but with reference to how this corresponds to the MIDI version of the same concept. This generality has already paid off in application adoption. The Notion program is one of the first symbolic music programs not to use MIDI internally for playback. It became the first program to support MusicXML ahead of MIDI, in part because MusicXML mapped more closely to Notion's key application concepts than MIDI did.

MusicXML represents musical pitch with a <pitch> element, measures with a <measure> element, and so on. Compare this to SMDL's usage of foreign terminology like cantus and gamut, or NIFF's focus on the

graphical elements of notation rather than the musical elements – so much so that NIFF includes no direct representation of musical pitch (Good, 2001).

Prefer Clarity over Concision

MusicXML takes section 1.1 of the XML specification seriously when it says that "Terseness of XML markup is of minimal importance." Names are fully spelled out. Musical data is represented using elements, while metadata such as formatting and performance information is represented using attributes (Harold, 2004). The clarity and flexibility of structure provided by multiple elements generally outweighs the benefit of a smaller XML file using attributes.

As MusicXML has evolved from version 1.0 to the current work on version 1.2, the importance of these decisions has become greater over time. There are always some cases where the element/attribute or data/metadata choice is unclear. I have never regretted choosing elements over attributes. It is the rare cases where attributes were ill-advisedly chosen over elements that have tended to cause problems during the growth and evolution of the format.

Roland's MEI format takes nearly the opposite approach, leading to a more concise encoding that is harder for software developers to work with. MusicXML's octave element corresponds to MEI's oct attribute; MusicXML's duration element corresponds to MEI's dur attribute. Data for duration and octave is carried over contextually in MEI rather than being spelled out explicitly in each note as in MusicXML. MEI-specific terms like pname and meiform show up in the simplest MEI examples. MusicXML files are longer than the corresponding MEI files, but they are more structured, complete, easier to understand without reading documentation, and easier for music application developers to work with.

Make the Structure Compatible with Leading Applications

MusicXML was placed on a strong technical foundation by building on the two leading academic formats for symbolic music: Hewlett's MuseData format (1997) and Huron's Humdrum format (1997). The first versions of MusicXML were basically an XML version of MuseData, with key concepts from Humdrum added in. Soon MusicXML added features for popular music and other areas that went beyond the original MuseData and Humdrum designs.

We knew from the outset that MusicXML was compatible with these academic formats. MuseData's structure was also designed to be compatible with MIDI output, to facilitate MuseData to MIDI translations. A key part of early MusicXML development in the first two years was to ensure that MusicXML's structure was compatible with the major commercial symbolic music applications. We found that MusicXML's basic structure was very well matched to the Finale and Sibelius data structures. This made it easier to build our own software for these applications, and gave us confidence that other music applications would find a similar good fit. This indeed turned out to be true in practice.

A major reason why NIFF was only adopted by music scanning applications is that its graphical orientation is compatible with those applications. But it is quite foreign to other music applications, making NIFF very difficult for most music software developers to work with. Sibelius's NIFF reader, for instance, had problems in reading simple pitch information correctly (Good, 2001).

Another reason why MusicXML is compatible with leading applications is that it primarily models a document – a musical score – rather than an abstraction of a document. The most well-known abstraction in the music area is SMDL's division of music into logical, visual, gestural (performance), and analytical domains (Sloan, 1997). As we discuss below, this is a useful framework for thinking about different aspects of music representation. It extends the popular idea of separation of presentation and content in a sensible way for music applications.

But strict separation of presentation and content can pose more problems than it solves. This is especially true in music notation, where presentation conveys semantic meaning to the user. The horizontal spacing of musical notes and marking conveys information about how music is performed over time. It is not "just" a matter of presentation. Most major music applications recognize these interdependencies and integrate the

different domains together in a single format, separating the different concerns into different data structures with varying degrees of modularity. MusicXML follows this approach: the content domains (logical and analytical) are generally represented in elements, while the presentation domains (visual and gestural) are generally represented in attributes.

Note that we are advocating making the structure compatible with leading applications. This is not the same as adopting application-specific terminology or design decisions. It is tempting to do this, especially in situations like ours where the development of the language was tied closely to the development of a translator for a single major application. But this temptation should be resisted to avoid biasing the format to one application. Such bias would make it more difficult for others to work with the format, both for technical and social reasons.

Develop the Format Together with the Software

Iterative design and evolutionary delivery techniques have been used since the 1980s to produce more usable and useful computer systems (Gilb 1988, Good 1988, Gould and Lewis 1985). With MusicXML, we have found that these techniques can also be successful in XML language design. By building software to use the language together with the language itself, we were able to discover and fix the technical problems that would impede adoption – either by making certain music exchange impossible due to gaps in the format, or by making it difficult due to suboptimal design and implementation decisions.

Since MusicXML was designed for interchange between many different types of applications, our first prototypes concentrated on making sure that we could handle applications that focused in very different areas. SMDL's concepts of logical, visual, gestural (performance), and analytical domains were helpful for this purpose. We built prototypes in Visual Basic 6.0 for each domain that was supported by commercial applications:

- Logical: MuseData to MusicXML translator
- Visual: NIFF to MusicXML translator
- Gestural: MusicXML to MIDI translator

We also built two analysis prototypes that worked off of data in the logical domain. These were small Visual Basic programs that showed note duration distributions on a histogram and pitch/duration correlations on a scatterplot.

Once we had successful implementation experience with these prototypes, we moved on to a deeper implementation experience with a real commercial application: a prototype translator from Finale's ETF format into MusicXML.

While MusicXML started being used in commercial applications with MusicXML 0.5 in 2001, version 1.0 did not ship until January 2004. This delay was to give us plenty of time with different applications so we could be sure that we would not need to make any incompatible changes to MusicXML 1.0 in a future release. MusicXML 1.1 is a strict superset of MusicXML 1.0, and likewise MusicXML 1.2 will be a strict superset of MusicXML 1.1. We feel it is crucial to preserve developer investment in their MusicXML software, and extensive experience with a wide range of music software was crucial to giving us the confidence that MusicXML was sufficiently mature and stable to guarantee this level of future compatibility. MusicXML 1.0 was released after the developer community included people working on diverse applications such as digital music stands, sequencers, algorithmic composition, and tablature editing – not just standard music notation editing and music scanning. By the time MusicXML 1.0 was released there were implementations on Windows, Macintosh OS 9 and OS X, and Linux, using languages such as C, C++, Visual Basic, Java, and ManuScript (a Sibelius-specific derivative of Simkin).

As mentioned earlier, the Macintosh operating system is very popular in the music market. This is especially true for high-end professional users who are natural early adopters. Yet most of our early software development was for Windows only, using the same mix of Visual Basic 6.0 and Visual C++ 6.0 that I had used in my previous position at SAP. Given the universal failure of past efforts at standard

symbolic music formats that went beyond MIDI, I wanted to minimize risk from extra factors such as switching development platforms and programming languages at the start of the project. The start of a high-risk project did not seem like a good time to learn a new development environment.

We also did not fully understand the extent of the Macintosh's dominance of the high-end music market until we demonstrated MusicXML at the industry's large NAMM trade show in 2002. This convinced us that we had to do a substantial rewrite of our production software from Visual Basic into Java. Our initial Windows-only approach maximized our chances of initial success, but then slowed adoption of MusicXML until our own software was cross-platform. We did get the important advantage of avoiding most of the development costs associated with the transition from Macintosh OS 9 to OS X. Due to the lack of a modern Java version on Macintosh OS 9, Finale's MusicXML support on Macintosh is for OS X only.

Support a Market Leader Early

When Recordare started the MusicXML project, a key milestone was that we be able to read and write MusicXML files from either Finale or Sibelius within a set period of time. If we did not accomplish that goal, we would give up and I would move on to other things. Both in 2000 and today, Finale and Sibelius are the two dominant players in the music notation marketplace. An interchange format that could not support one of these two applications would be of commercial interest to nobody.

On the other hand, once we could do two-way interchange to one of these applications, this opened up a wealth of possibilities to attract other software developers. Developing high-quality music notation display and editing is a very complicated task. By removing the need to duplicate what Finale and Sibelius provided, we would lower the barrier to entry for innovative new music applications. Of course, we would also be lowering the barrier to entry for competitive applications, so neither Finale nor Sibelius had much incentive to help us with this project. We hoped that they would see the benefit to them once we had working software, but we never expected these entrenched market leaders to give us help in our initial development work.

Both Finale and Sibelius provide plug-in development kits so that third-party developers can write add-on software to their applications. It quickly became apparent that only Finale's was capable of two-way interchange, so that is where we focused our efforts. Building a translator that supports major, feature-rich commercial applications at a reasonable level tends to be a substantial undertaking. We therefore held off on doing this until our experience with the initial prototypes satisfied us that our basic MusicXML design was on the right track.

Once we had a working plug-in for Finale that we were willing to alpha test outside Recordare, we started approaching third-party developers. Our first obvious target was Graham Jones, developer of the SharpEye Music Reader software. At that time, each notation editor had its own captive scanning program. Finale worked with the SmartScore family of products provided by Musitek, while Sibelius worked with the PhotoScore family of products provided by Neuratron. SharpEye Music Reader had developed a reputation for scanning accuracy equal or better than these programs. But it could only use MIDI to transfer music into the notation editors, negating the advantages of the program's internal scanning accuracy.

Graham Jones saw the opportunity, even though he was not happy about having to duplicate work that he had already done to export NIFF files. Since MusicXML is farther from the internal structure of scanning applications than NIFF is, it would be even more work to support MusicXML. But the benefits from being able to better sell to Finale users was worth the cost, and in September 2001 SharpEye became the first commercial product to support the MusicXML format. The makers of Finale then saw the advantage of having a scanner like SharpEye work better with their program. They started including MusicXML support based on Recordare's Dolet® software technology with the Windows release of Finale 2003. SharpEye was Windows only, as was the original version of our Dolet plug-in software, so built-in Macintosh support did not get added until Finale 2006. Universal binary support for Macintosh was added in Finale 2007.

Figure 1 reflects MusicXML's implementation status after SharpEye had shipped, but while our Dolet for Finale plug-in was still in beta test. The change from Figure 1 to Figure 2 shows the order of magnitude

increase in MusicXML support over the past 5 years. As we had thought, having MusicXML support for Finale was a key element in attracting other software developers to the format.

The next major challenge was to add MusicXML support for Sibelius. The original Sibelius plug-in development kit contained no way to interact with the computing world outside of Sibelius, such as through reading or writing a text file. The necessary features for MusicXML export were added in version 2.1, but not documented until version 3.0. We created a plug-in for Sibelius export soon after one of our customers directed our attention to this newly documented feature. The capabilities of the exporter increased as Sibelius increased the power of the plug-in environment to meet their own application needs. Sibelius added built-in MusicXML import support when Sibelius 4 was released in 2005. This was done primarily to import Finale files more accurately than had been possible with their own special-purpose Finale translator. This became feasible once Finale provided built-in MusicXML support on both Macintosh and Windows. MusicXML export in Sibelius 4 still requires our separate Dolet translation plug-in.

The music scanning world also adopted MusicXML more widely once SharpEye and Finale supported it. PhotoScore Professional added support for version 3 in 2003; Kawai SCOREMAKER for version 4 in 2004, and SmartScore for most editions of version 5 in 2006. Music scanning and notation programs can now be freely mixed and matched based on customer needs, rather than being restricted through vendor tie-ins.

It is difficult to underestimate the importance of having an XML interchange format support a market leader early in the process. If your so-called standard format does not support at least one of the most popular programs in the market, why should you expect anyone to adopt it? This support is the responsibility of those who want the XML interchange format to succeed. It seems naive or cynical to expect the market leader to do this for you. It is rarely in their business interest to invest time and money in doing so. However, once support is available independently, market leaders may find a way to turn it to their advantage, as happened with MusicXML. This dynamic is becoming more important today, as more and more people realize the importance of open file formats for long-term use and preservation of computer application data.

Fortunately, in many application areas the market leaders have a plug-in development environment at least as good as what was available to us in Finale 2000. This is exactly what is needed for format proponents to build their own support for the market leader. Proprietary file formats are rarely documented, but documented plug-in environments that can access the file format are much more common. Many of these plug-in environments are much more mature than what we had to work with to build our initial MusicXML plug-in for Finale. Our progress would have been far more rapid if we had access to a plug-in development environment as mature as the one that Microsoft provided as far back as Office 97.

Market to Other Software Developers

One place where prior efforts like NIFF and SMDL failed was in marketing. Once the NIFF and SMDL formats were defined, there was not a concerted, long-term effort to drive adoption in the marketplace. Adoption of a new interchange standard takes time, especially in a small market area like music notation.

The Internet provides an excellent channel for marketing to software developers worldwide. We regularly use the Web as well as print media to learn about software programs that we feel could benefit by adding MusicXML support. Our marketing efforts really began in October 2001 with the first MusicXML beta release. This was followed closely by the establishment of a MusicXML eGroup, later moved to Yahoo! Groups and then to a developer mailing list. We put the DTD files on the site together with a tutorial, example files, and links to MusicXML software and publications. We have attended the industry's major NAMM trade show in the USA since the company was founded. We also attend conferences and trade shows elsewhere in Europe and North America to spread the MusicXML message to all sorts of developers, be they in academia or industry, using closed-source or open-source models. We track all this activity in a summary spreadsheet which summarizes the MusicXML status or potential for hundreds of applications.

Sometimes it takes success in one product category before one can move to the next category. The major commercial sequencers are currently missing from the MusicXML implementation chart. MusicXML

support really only became commercially interesting to these applications once both Finale 2006 and Sibelius 4 had built-in MusicXML import on both Windows and Macintosh. Adding MusicXML support now makes sense to help the workflow of customers who compose in the sequencer but have scores and parts prepared in notation programs like Finale and Sibelius. The major sequencers are elaborate products with complex development schedules, but we hope it is not much longer before some major sequencers support the MusicXML format.

Part of marketing to developers involves removing barriers to entry. XML itself helps a great deal with that due to the widespread availability of free, high-quality XML parser software on just about every modern development platform imaginable. Our MusicXML licensing contributes to this via a royalty-free license modeled on the W3C license used for XML itself. This license was developed in the days before widespread concerns about license proliferation. To date we have not had issues with license compatibility, but this is something we will monitor during the MusicXML 1.2 cycle to see if a change to a more standard license might be appropriate.

Give Format Developers Good Support

Closely related to marketing to software developers is supporting them once they choose to use your format. We work to be very responsive to developers using the format and software, replying as quickly and accurately as possible to developer questions.

The MusicXML developer lists have provided an opportunity for discussions to happen at an open community level, with developers helping other developers. But many development activities must be confidential until a product is released, so we provide private e-mail support as well. This technical support is free for MusicXML developers. Trying to charge for either the format itself or format technical support in the music software industry is a sure path towards irrelevance. Recordare does offer consulting services for more complex tasks, including adding MusicXML support to third-party products, or adding features to Recordare's Dolet translation plug-ins that are needed for a particular project.

This active developer community is an enormous benefit for the MusicXML format. During the MusicXML 1.1 development process, many of the best ideas for how to solve complex problems came from MusicXML developers from all over the world, who need MusicXML solutions in order to take advantage of business opportunities for their applications.

Avoid Overhead

Since the MusicXML project started, we knew we would face some resistance because the format was owned, developed, and maintained by a single, small software company. Recordare joined both OASIS and the MIDI Manufacturers Association with the idea of networking with these industry standards communities, and possibly submitting MusicXML to a standards process in the future.

This activity took a big step forward in November 2003 when Recordare, Indiana University, and Matthew Dovey started an OASIS discussion list on music notation. As part of this discussion process, we met with as many industry members as we could at the 2004 NAMM show to gauge their interest and willingness to participate in the standards process.

We received very sage advice from our industry supporters, who strongly urged us to avoid the path of a standards organization and maintain the PDF model of an open standard supported and controlled by a single company. They counseled that those not yet using MusicXML were not doing so because of its provenance, but for their own business reasons. Moving MusicXML to a standards organization would not have any positive effect on its adoption. If anything, it would make things worse because MusicXML would not be able to adapt to changing circumstances as quickly as before. Based on this advice from our customers, we decided not to move MusicXML into a standards organization like OASIS.

The wisdom of this advice was amply demonstrated in 2005. Major business opportunities led to the rapid development of the MusicXML 1.1, with over 70 new features to better support music formatting and

digital sheet music preparation. These features needed to be designed, implemented, and tested very quickly, as MusicXML 1.1 had to ship sufficiently in advance of immovable product deadlines from three separate companies. This would never have been possible in the higher-overhead environment of a standards organization, even one as industry-friendly as OASIS. The MIDI Manufacturers Association provides a more lightweight process than OASIS, but typically at the cost of active participation by contributors from outside the music products industry.

Standards organizations work well for mature, well understood technologies. They provide a valuable organizational framework for complex negotiations between diverse stakeholders in a technology with a large potential market. Formal standardization in large markets can save money overall, and in life-critical applications it can save lives. But in small vertical markets like symbolic music software, these organizations are too costly in both time and money for even the largest industry members. (A large industry member in symbolic music software is one that has a developer staff in the double digits.) It is particularly costly given that, by itself, the backing of a standards organization is no guarantee that a format will become more widely used.

The lack of relevance of standards organizations for music notation software was demonstrated by SMDL, which was never implemented by any commercial product despite the ISO 10743 seal of approval. History appears to be repeating itself with MPEG's recent project on Symbolic Music Representation, which is busy integrating WEDELMUSIC-based technology into MPEG despite a near-unanimous show of industry non-support.

Getting an XML format adopted as an interchange standard is a social and technical process. For the largest, most well-understood areas, standards organizations can help in both processes. But for smaller industry areas like music notation, the overhead of standards organizations can hinder these processes, rather than help them.

Many factors aid the adoption of an interchange standard. It is very important for the user community to have an active voice in the development and maintenance of the standard – a voice that is truly powerful, not simply for show. But there are many means to that goal, and individual companies can provide a more effective mechanism than standards organizations in some circumstances.

Conclusions

MusicXML has succeeded in becoming the de facto interchange standard for symbolic music applications that use common Western music notation. It succeeded in this task where numerous previous and contemporary efforts failed. I hope that this analysis into factors that seemed particularly important in MusicXML's success help other people to drive the successful adoption of open interchange standards in other application areas. Standards adoption is a technical and social process, and careful attention must be paid to both to meet with success.

Acknowledgments

Walter Hewlett, Eleanor Selfridge-Field, David Huron, and Barry Vercoe provided the most important prior art and ongoing encouragement for the early development of the MusicXML format. Curtis Morley and the musicRAIN team made major contributions to the MusicXML 1.1 project. Geri Actor, Doill Jung, Graham Jones, Didier Guillion, Mark Maronde, Don Byrd, Christof Schardt, Donncha Ó Maidín, Bernd Jungmann, and the many members of the MusicXML mailing list have made great contributions to the MusicXML community over the past seven years.

References

Bellini, Pierfrancesco and Paolo Nesi (2001). WEDELMUSIC Format: an XML Music Notation Format for Emerging Applications. In *Proc. First International Conference on Web Delivering of Music* (Florence, November 23-24), pp. 79-86.

Castan, Gerd, Michael Good, and Perry Roland (2001). Extensible Markup Language (XML) for Music Applications: An Introduction. In *The Virtual Score: Representation, Retrieval, Restoration*, ed. Walter B. Hewlett and Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 95-102.

Dumitrescu, Theodor (2001). Corpus Mensurablis Musice "Electronicum": Toward a Flexible Electronic Representation of Music in Mensural Notation. In *The Virtual Score: Representation, Retrieval, Restoration*, ed. Walter B. Hewlett and Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 3-18. Also see http://www.cmme.org/.

Gilb, Tom (1988). *Principles of Software Engineering Management*. Reading, MA: Addison-Wesley.

Good, Michael (1988). Software Usability Engineering. *Digital Technical Journal*, No. 6, 125-133. Republished at http://www.recordare.com/good/dtj.html.

Good, Michael (2001). MusicXML: An Internet-Friendly Format for Sheet Music. In *Proc. XML 2001* (Orlando, December 9-14), http://www.idealliance.org/papers/xml2001/papers/html/03-04-05.html.

Good, Michael (2006). MusicXML in Commercial Applications. In *Music Analysis East and West*, ed. Walter Hewlett and Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 9-20.

Gould, John D. and Clayton Lewis (1985). Designing for Usability: Key Principles and What Designers Think. *Communications of the ACM*, 28 (3), 300-311.

Grande, Cindy (1997). The Notation Interchange File Format: A Windows-Compliant Approach. In *Beyond MIDI: The Handbook of Musical Codes*, ed. Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 491-512.

Harold, Elliotte Rusty (2004). *Effective XML*. Boston: Addison-Wesley.

Hewlett, Walter B. (1997). MuseData: Multipurpose Representation. In *Beyond MIDI: The Handbook of Musical Codes*, ed. Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 402-447.

Huron, David (1997). Humdrum and Kern: Selective Feature Encoding. In *Beyond MIDI: The Handbook of Musical Codes*, ed. Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 375-401.

Huron, David (2006). *Sweet Anticipation: Music and the Psychology of Expectation*. Cambridge, MA: MIT Press.

MIDI (1996). *The Complete MIDI 1.0 Detailed Specification*. Document version 96.1. Los Angeles: The MIDI Manufacturers Association. Also see http://www.midi.org.

Roland, Perry (2002). The Music Encoding Initiative (MEI). In *Proc. First International Conference MAX 2002: Musical Application Using XML* (Milan, September 19-20), pp. 50-55. Also see http://www.lib.virginia.edu/digital/resndev/mei/.

Selfridge-Field, Eleanor, ed. (1997). *Beyond MIDI: The Handbook of Musical Codes*. Cambridge, MA: MIT Press.

Sloan, Donald (1997). HyTime and Standard Music Description Language: A Document-Description Approach. In *Beyond MIDI: The Handbook of Musical Codes*, ed. Eleanor Selfridge-Field (Cambridge, MA: MIT Press), 469-490.

Steyn, Jacques (2002). Framework for a Music Markup Language. In *Proc. First International Conference MAX 2002: Musical Application Using XML* (Milan, September 19-20), pp. 22-29.