MathML is a failed web standard

26 Mar 2016

Since I'm expecting more than my ~~1~~ 2 regular readers to read this, let me add a preface. I've been managing the MathJax project for 4 years now. In the last two years, I've also been consulting for publishers regarding math-related workflows, in particular TeX-to-XML and XML-to-web, both on the back and front-end. Not that it says much, but I'm an invited expert on the W3C Digital Publishing Interest Group (shoutout to Jean and Tzviya!) and I also recently left the W3C Math Working Group (to which David had invited me in late 2014). MathML is a big (positive) part of my professional life.

I recently posted a terse – uhm, shall we say summary? – of my thoughts on MathML on a11ySlackers; and I promised a blog post. There's now a 6000 word thingie sitting in my drafts which would take months to whip into shape. So I tried again and it now feels both too long and too short; oh well, maybe it leads somewhere, maybe it doesn't.

Needless to say, opinions posted on my personal website are my personal opinions (funny how that works). In particular, they do not reflect the opinions of any of my clients, let alone the team at MathJax. I think they don't particularly help anything or anyone specifically except, perhaps, in encouraging a more open and realistic discussion.

The gist of it

MathML is a failed web standard.*

We can do better, we deserve better.

MathML-in-HTML5 is in the way of that.

*Some people might prefer "browser standard", as in "a web standard to be implemented natively in the browser" since some web standards do not rely on browser implementations. Also, "natively" as opposed to some web-components hack shipped in a browser.

MathML is a failed web standard

It doesn't matter whether or not MathML is a good XML language. Personally, I think it's quite alright. It's also clearly a success in the XML publishing world, serving an important role in standards such as JATS and BITS.

The problem is: MathML has failed on the web.

Luckily, many technologies have succeeded and today MathML is neither necessary but also no longer sufficient for math on the web. Instead of one monolithic solution, we have many. We should acknowledge this and move forward towards several newer and smaller standards that actually help developers.

Here are a few reasons that make me say these things.

1. MathML has not been significantly supported by browser vendors, neither on the spec level nor on the implementation level.

You might easily think they do (Office! ChromeVox! VoiceOver!) but the browser vendors actually don't. The partial MathML implementations in Gecko and WebKit are entirely the work of volunteers. Largely unpaid, largely unsupervised, largely unaccountable.

Not a single browser vendor has stated an intent to work on the code, not a single browser developer has been seen on the MathWG. After 18 years, not a single browser vendor is willing to dedicate even a small percentage of a developer to MathML.

This is where the story should end, really. But sadly it doesn't. MathML's success in the XML world has kept it alive, but not for the benefit of anyone on the web.

MathML is a poor web standard and it would be better to remove it from HTML 5.

2. Browser implementations of MathML are not used.

If you look at publicly available crawler data, you'll notice that it's hard to find examples of MathML that aren't behind paywalls. If you look further, you'll hardly find an example where people providing MathML content rely on native MathML implementations; even on Gecko and WebKit they use MathML-to-HTML5 converters. Another indicator is that, despite implementations having subtly deteriorated in the past two years, people aren't even complaining (I mean, WebKit stopped drawing surds (try this in Safari 8) but apparently nobody cared enough to even file a bug). Actual developer problems are so extreme you can't seriously develop anything slightly advanced with MathML (e.g., Gecko has non-existent or incomplete support for basic APIs such as style, dataset, or event handlers for MathML elements).

3. Content MathML has failed to provide usable semantics.

Ok, truth be told, I don't know. The problem is: it's nearly impossible to generate good Content MathML (except with massive manual labor). As far as I know there is not a single significant collection of mathematics encoded in Content MathML out there. It's mainly ephemeral research projects and some hand-crafted projects. That's fine, we need research after all, but that is not a

standard fit for the web.

4. Presentation MathML fails front-end developers because it unnecessarily binds layout features to MathML elements instead of providing a usable set of CSS features.

Now `<mstyle>`, `<mspace>`, `<mpadded>`, `<mphantom>`, `<menclose>`, `<mfenced>`, `<mtable>`, `<mstack>` might sound funny to a web developer but it's a serious problem. The web has found a productive separation of concern. MathML is incompatible with this approach.

5. MathML fails because it does not specify layout sufficiently.

MathML assumes an implementor would know or care about the intricacies and traditions of math layout. How do you draw a surd? Not specified. How do you draw a fraction? Not specified. How do you space things? Not specified. [But yes, dear implementor, you should support arcane mathematical layout features like movable limits, operator dictionaries, the subtle spacing and layout difference of inline- and display-style and so forth; you know why they're important, right? RIGHT? And also make sure to implement 5 different approaches to vertical stacking, because, reasons – kthx, xxo.]

6. Presentation MathML fails because CSS is slowly implementing all layout features that mathematical layout needs, making it obsolete.

Today, lots of tools will let you render mathematics using CSS. It's messy but it works everywhere (ok, dear IE7 user, not for you, I'm sorry). The time when MathML implementations would have significantly enhanced web layout features are past.

7. Presentation MathML fails to provide sufficient semantics.

Neil Soiffer wrote ingenious heuristics for MathPlayer which makes most people think that Presentation MathML makes mathematics accessible. That's about as accurate as saying OCR means all

images with text are actually accessible.

The reality is that even for school-level math you need both high-quality Presentation MathML (which is rare in itself) combined with powerful (but inevitably fallible) heuristics to extract meaningful semantic information; that's acceptable in the short run but not a real solution for mathematical semantics on the web.

8. The MathML spec is not actively developed.

MathML has seen no significant activity in almost a decade. In the industrial XML world, MathML is a success and people want more features but improvements are not even brought up. It seems nobody wants to jeopardize an adoption on the web. MathML being a web standard is negatively affecting even those users who actually embrace it because MathML is stuck in maintenance mode.

Did you know the MathWG's charter is running out this month? Would you notice if it wasn't renewed and the WG would cease existing? Would you notice if WebKit and Gecko ripped out their MathML implementation tomorrow? I'm not sure many people would.

What to do next.

A) MathML needs to be dropped from HTML 5.

Many people I've met have the mistaken impression that browser manufacturers have declared an intent to implement everything in the set of standards usually called HTML 5. They have not (even if HTML 5 as a "spec" may strive for that).

I think as long as MathML is in that set of standards, the lame duck argument ("it's a standard!") will continue to prevent alternative developments that help the actually working solutions for

mathematics on the web.

At this point, MathML is effectively preventing mathematics from aligning with today's and tomorrow's web. This is hurting everyone. We need to drop MathML to make room for better standards.

B) Math layout can and should be done in CSS and SVG. Let's improve them incrementally to make it simpler.

It's possible to generate HTML+CSS or SVG that renders any MathML content – on the server, mind you, no client-side JS required (but of course possible). The resulting markup is arguably crap – it's span soup at its worst and some use cases are difficult to realize. But we've been there with HTML and CSS; people know how to solve this. It got us standards like flexbox and css-grid; it's worth pursuing improvements to those standards that work instead of waiting for Godot.

It's also difficult to write your own math rendering tool. But we need more ideas, not less! It shouldn't be harder to write a simple math renderer in CSS or SVG than it is to write a RWD framework or a vector graphics library.

We don't need Presentation MathML for this even if many projects (like MathJax) use it as an internal format. MathML's failure as a web standard is hurting the web because it is blocking discussions about improving existing standards to help existing mathematics tools on the promise that eventually "MathML will solve everything (tm)".

I can't see a native MathML approach help to fill these final gaps. What existing rendering solutions need has little to do with what MathML implementations need. We don't need underspecified layout features tied to MathML elements, we need flexible CSS features that are integrated into existing CSS. Most importantly,

existing solutions can iterate on partial improvements to ensure that these help layout on the web more generally, not just the needs of one specific mathematical markup language.

We don't need one true approach to math layout, we need flexibility for developers to be innovative and pursue new ways of solving layout problems and expressing mathematical thought on the web.

We need to get together with CSSWG/Houdini TF/etc to work out solutions that help those developers who actually solve the problem of math on the web.

To give a rough idea – From a MathJax point of view, three areas are difficult in CSS right now (and probably universally for math layout tools on the web):

- vertical stacking (although flexbox is probably already enough to fix that and CSS Ruby might also be interesting to look at for synergies)
- stretchy characters and enclosures (this is the BIG one – but they're really just fancy borders)
- tight character bounding boxes (math layout has stronger requirements on typography than most forms of text but even existing technology is problematic)

Stretchy things are by far the biggest layout question, if only because they once led Ojan Vafai to call math layout fundamentally incompatible with CSS layout. As much as I respect his expertise, that cannot be the answer. It seems unlikely that we can't incrementally reduce the complexity for existing rendering solutions; in any case, it has little to do with MathML.

C) We need a new approach for exposing semantics.

Since layout is practically solved (or at least achievable), we really

need to solve the semantics. Presentation MathML is not sufficient, Content MathML is just not relevant.

We need to look where the web handles semantics today – that's ARIA and HTML but also microdata, rdfa etc. Especially ARIA is an extremely urgent problem because it currently ties mathematics entirely to Presentation MathML elements (where it fails) instead of providing a way to enrich all mathematical rendering on the web.

We also need to look beyond the semantics of mathematics into the semantics of mathematics in its applications, e.g., mathematical notation out of physics (units etc), chemistry (isotopes, reactions etc) and biology (trees, graphs etc). We need to find ways to expose this information to assistive technologies, search and other tools.

To leave a comment write me an email.