Meinte Boersma

Follow 177 Followers About

Post-mortem for Más

Meinte Boersma Jan 31, 2018 · 7 min read

Más was my attempt at a Web-based workbench for projectional modelling languages which I've effectively abandoned late 2014. You can still try it out at the link given, although you'll have to use a non-WebKit browser such as Firefox: the JavaScript happens to rely on some API which has been patched shut in WebKit.

A short history of Más

In 2011, I attended the CodeGen conference with the co-located Language Workbench Competition (LWC). My key takeaways from the latter were:

1. There are *plenty* of language workbenches out there. (In fact, to date every edition of the LWC has seen a small set of recurring participants, such as MPS and the Whole Platform, but many more new, and often short-lived, entries.)

2. None of these were Web-based in the sense that they produced languages and environments for them that lived in the browser. (The remarkable thing is that almost 7 years later the situation's hardly improved: we now have DSL Forge for textual — i.e., non-projectional — languages, and that's really about it… We do have Web-based IDEs as well as the Language Server Protocol, so things might certainly improve.)

I decided to scratch this particular itch myself, and started with Más right after the LWC. Más stands for Model-as-a-Service, and I envisioned it as that: a one-stop shop in the form of a Web application that catered for all your Web-modelling needs which allowed you to develop and implement your language as well as have your language users do their actual modelling with. Think of Lucidchart but then with a meta-side as well. Similarly, I was aiming for a subscription-based business model to monetise my efforts.

Almost a year after starting on it, it was “done” enough to demonstrate it at the LWC-part (where the C now stood for the less competitively-named “Challenge”) of the CodeGen conference. This was heaps fun, but unfortunately didn’t bring in any actual customers. It seemed that attendees liked the product but couldn’t see themselves using it in an actual project. This already pointed to a fundamental flaw in the service — on which more later — , which I only realised to its full extent much later on — in hindsight, I was simply in denial about it.

After having taken on various freelance projects throughout 2012 to 2014, I decided to discontinue working on Más and instead join Mendix to help them create their Web Modeler which was finally released in 2017. In that sense, Más was successful since it had garnered me experience with and a reputation for being able to move modelling environments to the Web.

I had several direct reasons to stop work on Más:

1. I finally fully grasped the fundamental flaw in Más: *no* clear/easy/simple/conventient *way to integrate it within a software project* — more on that below.

2. I was starting to regard a modelling environment as “just another application to manipulate data”, as epitomised by this blog. This blog post argues that we primarily need a good programming language/paradigm for data-driven, UI-rich applications, and that such a “Generic Application Language” would make language workbenches to some extent superfluous since you’d “just as easily” could create a modelling environment directly in GAL code. This was also one of the main drivers for me to join Mendix, since their visual modelling language and environment certainly strives to be a GAL.

3. I was using Martin Thiede’s Concrete editor which was proving to be insufficient for what I wanted to do with it.
Mainly this was because Concrete is many things at once: a model framework akin to EMF, a rendering engine for such models, as well as the actual user interaction. This turned out to be hard enough to integrate and use myself for the one language (the Language Definition Language; LDL, for short) that Más has built-in, and even

harder to parametrise so that language developers could define their own projections. This meant that language builders would've needed to inject HTML+CSS+JavaScript in a way which was going to be anything but declarative, to build languages with customised layout and interaction.
I would have to disentangle Concrete into several separate concerns to allow myself to move forward, but that would have cost me a lot of effort.

Pitfall 1 — no integration

As already advertised: it took me a long while to come out of denial on this one…

Basically, the only way to do something with something modelled in Más (in a language modelled in Más) was to download a JSON or XML representation of it. I was a long way off from allowing language builders to define generators in Más, also because of the editor framework that was holding me back. But even that would not have made enough of a difference to swing the pendulum, since I also didn't have proper versioning, etc.

I alluded to this problem in a previous blog as well, and I find Más is not the only language workbench suffering from it. My advice: envision how to make your language workbench integrate with a well-chosen subset of types of actual projects in terms of both audience (see also pitfall 4), technology (platform, IDE, versioning, etc.), as well as process (collaboration, issue tracking, releases, etc.). Let this vision drive the rest of the effort, and you just might end with a usable or even sellable product.

Pitfall 2 — not picking the most boring technology

This is a general thing for startup-type software — see e.g. Zef Hemel's blog. Either you're building a pet project to get you up to speed with new and cool/hot technologies, (exclusive-)or you're building a product in the most efficient and effective manner possible, so you can spend time on working towards a market fit, traction, and hopefully paying customers.

I fell into this trap by choosing the Google Application Engine which was relatively new at the time. I (maybe predictably) had problems getting the persistence part of it to work for me: you know, *eventual* consistency and all… I should have moved to Google's MySQL offering as soon as it became available.

So, stick with what you know well and feel comfortable with, and focus on shipping a product, listening to beta users, and finding out you have to almost start over with your code base anyway.

Pitfall 3 — meta-circularity is overrated

The fact that you can implement your piece of software using the same software might be a strong show for the likes of a GPL, but I don't see any real use for it. Not only does it make for raging meta-headaches and ensuing low productivity because it makes your code base the solution to an intricate fixed-point problem…right until it fails to do that, and you have to break out your Git-fu to be able to go anywhere.

Also: which mainstream GPLs actually admit a meta-circular implementation?

More importantly, the meta part of a language workbench might not be the most representative of workbenches. After all, *a language workbench serves to produce modelling environments with languages that capture the domains of your paying customers*. These languages probably don't look a lot like the languages which allow to define them in themselves — of course, they might share some patterns but that's probably it. By trying to be meta-circular, you might be entirely missing the type of product that you can actually make money with.

I initially started going down the path of meta-circularity with Más by defining the LDL in itself, until I realised it was only costing me, and leading me nowhere. So, I made a very small DSL to define some aspects — structure, some constraints — of my LDL with, so I could generate some of the code for its implementation. This made me much happier, and more importantly: much more productive.

Advice: keep things simple and do not go for meta-circularity. You might not be convinced now, but you'll thank me later.

Pitfall 4 — selling to developers

And I'm not the only one saying this. Selling to "the business" is much easier — almost ridiculously so. Thing is: you have to also convince the IT guys and girls servicing the business guys and girls that (1) your product is working, and (2) is not making them have a more difficult job, nor obsolete. After climbing that hill, you'll still have to convince the people that hold the actual bags of money and fountain pens to bank on your tiny company providing such a new-fangled Cloud service™ and trusting that you're still in business in a year time.

Más tried to cater for two audiences at the same time: language engineers building a Web-based, projectional language and domain experts using said language. So, I tried to avoid this particular pitfall by making the meta-part less developer-friendly and thus less sellable. In a way, this is a milder variant of meta-circularity pitfall. In hindsight, I should have split the product for these two audiences.

My €0.02: aim to provide business value to the business. That's not a guarantee to anything, but relying on developers to champion your product to the powers-that-be well enough for the latter to give you money is guaranteed to *not* happen.

Conclusion

Am I sad I abandoned Más? Yes, since I really believed in it, and I would've liked to see it get traction. I still believe firmly in the usefulness of a Web-based, projectional modelling environment, so it's also sad to see only little activity in that area.

Regardless of all that, I learned a lot building Más which has served me well ever since. Probably the most useful thing I learned is to really strive for an MVP, e.g. through sensible feature culling: you're much more inclined to actually "kill your darlings" (in the form of fanciful ideas about what should *absolutely* be in your product) if those darlings are burning your own money! I found I'm much keener to do the Lean Startup thing than even e.g. Product Owners.

I hope this post-mortem was useful for you. Don't hesitate to share your thoughts in the comments.

Programming PostMortem Meta