Thylacine work plan

Evan Kirshenbaum

*Wednesday, March 22, 2023*

With a new contract in place, I figure its worth taking a bit of time to go over what tasks I see ahead on the software side of the effort and to recommend where the focus (or, rather, foci) should be.

Assumptions

Here are a few assumptions I have going in:

1. **The DMF work will continue at HP.**

It actually looks to be very promising technology, and especially when it can be made to scale up a fair bit, it seems that it should be a an important and widely used technique.

1. **The effort is doomed without software support.**

I think that the past couple of years have demonstrated that without domain-focused software support

1. it's essentially impossible to even figure out whether the hardware works,
2. it will be very difficult (and, for non-programmers, nearly impossible) to use the hardware to do anything non-trivial.
3. it is much more difficult to explore and demonstrate what would be possible when the hardware becomes more functional and as it evolves.
4. **I’m not going to be around forever.**

I was retired when Keith talked me into coming back to help you guys out, and while it has been a lot of fun (mostly), retirement does have a lot going for it. I turn 60 next summer, and I suspect that that may be as good a deadline as any.

What this means is that it will be important that whatever code is there when I leave will need to be able to be maintained by others and modeling new and evolving hardware will need to be made significantly easier.

1. **This is impossible.**

The work outlined here would be a decent load for a team of, say, three or four software engineers, even if you ignore (as I have so far) any notion of rigorous systematic testing.

I’m good, but even I’m going to have to prioritize, and some things are likely going to not get finished or not be done in as timely a manner as would be desired. This would be the case even if I were allowed to not worry about hours and push through on tasks when I have momentum, but if my hours are going to be severely limited, it’s both going to make that much harder and mean that I’m going to be wasting time reestablishing context on what I was working on the prior week before I hit my limit. Also, the more context switching between tasks, the longer each task will take.

Focus areas

I see three areas that I will need to fucus on. With incomplete lists of associated activities, they are

1. **Supporting current activities**
2. As people continue to use the platforms, both internally and at universities, there will be a need for rapid turnaround on bugs that are identified and (possibly less rapid) addition of new features (or exposing features already present in the Python model). Similarly, users will need questions answered, macro code critiqued, code examples written, and debugging help.
3. The macro language[[1]](#footnote-1) (which has evolved from a complete kludge to something almost usable) still has a fair number of features that need to be implemented to turn it into something that can be easily used by non-programmers to run non-trivial protocols. Also, some of the higher-level activities (e.g., multi-drop mixing and dilution, thermocycling) are supported in the Python code but are not yet exposed in DML.
4. One of the hardest parts of specifying protocols currently (in both Python and DML) is getting drops (which may be moving in parallel) to where they need to go. There is very primitive traffic control that keeps them from bumping into one another, and the Python layer has facilities for manually keeping them from deadlocking, but the process needs to be made much simpler at both levels.
5. Ideally, as was in the plan from the beginning, it should be possible to simply specify the high-level operations that need to be performed (e.g., mixing, diluting, heating, resting) and the desired quantities and parallelism and let the system determine the actual placement on the board and necessary operations to satisfy it. This would be a significant improvement, but would also be a fair amount of work at the Python level and involve creating a new language (a “DMF Protocol Language”) to allow the specification.
6. The platform doesn’t yet support sensors, and there is only partial support for pipettors. (In particular, the Opentrons model still doesn’t actually support driving the robot to the actual locations on a Bilby and there’s no support for dealing with the fact that it doesn’t really handle tiny drops.)
7. Supporting the TFT-based large array DMF boards is going to require additional facilities.
8. **Enabling long-term maintainability** (aka “Preparing for life after Evan”)
9. As it stands, doing things like changing where magnets, holes, dead pads, and sensor targets are requires editing the Python code for the board implementation, along with the concomitant distribution hassles. To simplify this, I intend to design and implement a ***Board Description Language* (BDL)** that will allow non-programmers to describe and tweak the layouts of DMF hardware.

This will place the task of describing boards (or, more often, tweaks to boards) in the hands of the end user, who can simply create or edit a file that is specified when the system is run. This will also make it easier to switch between multiple versions of the same hardware.

1. With the addition of the BDL to describe arbitrary boards, half of the work of getting a new type of board implemented (at least as far as the software is concerned) is taken care of. The other half can and should be relegated to a ***Board Communication Abstraction Layer* (BCAL)**, which will need to be designed and implemented. This will allow a Python programmer to implement a module that takes a BDL description and specifies what it means to communicate with the described board using an Opendrop controller, using Glider (HAL), by passing a bitmap, or whatever method is thought of in the future. A default provided model will allow any BDL-specified board to be simulated and graphically displayed.
2. The current GUI has been known to be horrendously inefficient for some time. It needs to be replaced. I’m not sure I’m the person to do this, but what I can probably do is add a ***Board Visualization Abstraction Layer* (BVAL)** that will allow the right person to focus solely on describing how to provide the needed display and user-input (e.g., “put a rectangle here”, “change the size and color of this circle”, “call this method when the mouse is clicked on this pad”) using a given graphics package and leaving the actual interaction model to the core.[[2]](#footnote-2)
3. There is a *lot* of documentation that needs to be written. This includes
4. users’ manuals for DML (the macro language), BDL, BCAL, and BVAL as well as for writing protocols in Python.
5. an architecture document for the core Python code, including a description of the (very-non-traditional) future-based program­ming that underlies much of it and the threading model as well as how to add support for new pipettors and sensors. This will be necessary if somebody other than me is going to try to maintain the core code.
6. an architecture document for the DML (and, later, BDL) implementation, to explain how things get added to the languages.
7. The most promising approach to getting long-term support and develop­ment on the platform is likely to be to release as much of it as possible under an open source license and move it to a public platform such as the public GitHub. This will mean that non-HP people (e.g., university collaborators) will be able to maintain and add to the code base.[[3]](#footnote-3) This process would involve
8. determining which pieces you’re willing to release to the public and which ones you want to keep confidential or proprietary.
9. identifying an appropriate license under which to release the code. This can be trickier than it seems, as you need to ensure that HP will still be able to use the code commercially. (Some licenses taint things they’re used with.)
10. setting up the public repository and migrating over existing issues.
11. identifying the process by which external users are allowed to modify the repository. Typically, this will involve identifying a set of users who are allowed to accept pull requests.
12. **Assisting in ongoing design**

In addition, I would think that my expertise would be needed (or at least useful) in helping guide the design of future boards and more complex protocols. In particular,

1. figuring out how one can most effectively work with very large boards, especially those without dedicated wells. This will involve both deter­mining what the software primitives might be, consulting on how to use them in protocols, and advising on how that impacts the hardware design.
2. helping to design large-scale pipelined protocols and advising on hardware design that enables their implementation.
3. in general keeping abreast of things the hardware designers are proposing and speaking up when it looks as though they may make the device difficult to use from a software standpoint.
4. suggesting hardware features that would simplify the process of writing protocols.

1. “The macro language” is getting to be an increasingly poor description of what it is, let alone an actual name, so from now on, I’m going to refer to it as **DML**, which you can think of either as “Digital Microfluidics Language” or “DMF Macro Language”. [↑](#footnote-ref-1)
2. With this in place, I may actually be able to experiment with doing this using more efficient packages. [↑](#footnote-ref-2)
3. Of course, it also means that your competitors will be able to take advantage of it. [↑](#footnote-ref-3)