Goals for Thylacine, a language suite for Digital Microfluidics Platforms

Evan Kirshenbaum

*Wednesday, March 24, 2021*

Introduction and Paper Goals

In this informal paper, I will try to accomplish two things. In the first section, I will give my current understanding of how the HP microfluidics platform works, at least from the point of view of a user who wants to run experiments on it. That is, I will not be concerned with specifics of how the platform is built, but rather the capabilities it provides. Note: I fully realize that my understanding will be wrong, and I am counting on readers of this paper to point out my misconceptions. I am neither a chemist nor a mechanical engineer, and I have only recently become at all familiar with the notion of digital microfluidics. I will also try to mention directions it seems likely that such platforms will evolve in the future. (These will also likely be wrong.)

In the second section, I will sketch out my current vision for a suite of languages and associated software for running such platforms. In keeping with what appears to be the current local naming convention, I’m tentatively calling this suite *Thylacine* after the Australian Tasmanian tiger.

10,000-foot view of Thylacine

The core of Thylacine is the design of four languages:

* **MPDL**, the *Microfluidics Platform Description Language*, is a language for describing and characterizing specific DMF platforms.
* **MPAM**, the *Microfluidics Platform Abstract Machine*, is a low-level model (and associated language) for operations to be performed on DMF platforms characterizable by MPDL.
* **MRL**, the *Microfluidics Recipe Language*, is an intermediate-level programming language for specifying DMF recipes independent of a particular DMF platform.
* **MPSL**, the *Microfluidics Protocol Specification Language*, is a high-level (e.g., near-natural-language) specification language for specifying microfluidics protocols close to the way they are described in the literature, with little or no programming expertise required

To support these languages software tools will be designed and implemented:

* An **MPAM API** allows MPAM operations to be initiated from programming languages such as Java, C++, and/or Python
* An **MPAM emulator**, likely graphical, implements the MPAM API and can emulate DMF platforms specified by MPDL descriptions.
* An **MPAM interpreter** reads an MPAM program and executes it by making calls to the MPAM API
* An **MPAM assembler** converts MPAM programs to equivalent programs in other programming languages
* An **MRL** compiler takes as input an MPDL description and one or more MRL recipes and outputs an MPAM program that impelemnts the recipes for the platform
* An **MPSL** compiler takes as input an MPSL specification and translates it into an MRL recipe (and, optionally, from there to an MPAM program).

There will also be MPDL descriptions and MPAM API implementation libraries for various concrete DMF platforms

Caveat

This is, obviously, a lot of work, and there is, at present, only one of me, so it may well not all get done. I do, however, think that I see how to do all of it (even the MPSL compiler), so for now I’m going to work under the assumption that it is all possible.

My DMF Platform Understanding

In this section I will present a (flawed) description of digital microfluidics (DMF) platforms in general and the current HP prototype in specific. As stated above, I will be primarily concerned with the capabilities the platform presents to the user.

I should note up front that at the time I’m writing this, I know almost nothing about IonTouch or what it means for DMF platforms. As I learn more, I will revise this section (and likely my design choices for the Thylacine languages).

What is a DMF platform?

At its most basic level, a DMF platform presents an array of *pads,*[[1]](#footnote-1) each of which can contain a drop of fluid. In the current platforms, the basic drop size is 0.5 µL, and each pad can hold a volume of several drops. In some cases, some of the pads may be *unusable* (e.g., because that area contains a support for the board’s cover) and must be routed around.

Electrodes and drop motion

Integrated into each pad is an *electrode*. When a pad’s electrode is on, fluid in neighboring pads is enticed to move to the pad. This can be used in several ways:

* Most basically, it can be used to make a drop move from pad to pad across the board.
* When an empty pad is between two drop-containing pads, turning on its electrode attracts both drops and causes them to mix.[[2]](#footnote-2) My current understanding is that the mixing happens essentially immediately and completely.
* Finally, if neighboring pads to a drop-containing pad turn on their electrodes at the same time, the drop will be split between the neighboring pads. My understanding is that the drop is split into essentially equal parts, or close enough to equal for practical purposes, as long as the neighboring pads are across a pad edge from the drop-containing pad (as opposed to being neighbors across corners of the pad[[3]](#footnote-3)). I presume that if the electrode of the drop-containing pad is also on, that pad also participates in the splitting and retains a portion of its initial volume.

The DMF board has a *clock rate* that specifies the frequency with which droplet motion events can take place. In the current platform this rate is 10 Hz, meaning that it takes 0.1 seconds for a drop to move (or split) to a neighboring pad, and one clock cycle is sufficient for all operations regardless of the volume or content of the drop.

Electromagnets, magnetic beads, and drop separation

Some or all pads may have *electromagnets* underneath them, which may be turned on or off. If a drop contains substances bound[[4]](#footnote-4) to *magnetic beads* and the electromagnet under its pad is turned on when neighboring pad entice the drop to move, the beads—and, therefore, the bound substance—remain behind and may be transferred to another drop that later occupies the pad when the electromagnet is turned off.

Thermal regions

Some regions of the pad array have the property that their temperature can be controlled. By remaining in such regions for a number of time steps, the drop may be heated to a desired temperature. My understanding is that the regions are thermally isolated to the extent that pads next to thermal regions do not have to worry about the temperature.

Wells

Along the edges of the board are some number of *wells* (or *reservoirs*), each of which can contain a (relatively) large volume of fluid. The current roadmap posits between 32 and 96 wells. In the OpenDrop system, wells can contain 120 uL. In the roadmap, the goal is for most wells to contain 250 µL, with a subset able to contain 2 mL. Each well is associated with an *entry pad* on the one of the edge rows or columns of the board.

At least on OpenDrop, the well itself has a number of electrodes (five) that can be used to either dispense a drop from the well to its entry pad or to remove a drop from its entry pad to the well. If I understand the logic,[[5]](#footnote-5) it looks as though it should take three or four time steps in order to perform either operation.

Wells have the property that when they dispense their entire content, there is no residue left behind, so the well me be reused to hold drops of another fluid without worrying about contamination.

Although the current platforms do not yet do this, it is considered desirable that wells be able to perform operations that are more efficiently performed on volumes of fluid larger than a drop. The most likely such operation is *incubation*, in which a well may be heated to a desired temperature and held at that temperature for a specified period of time. This would be considerably more efficient than the current paradigm in which incubation is performed by walking drops through thermal regions for the desired amount of time. (It would also free up those pads for other use.) The roadmap specifies that wells will also be able to cool to 4 C, and perform “magnetic actuation” and “fluorescence quantification”.[[6]](#footnote-6)

It’s not clear to me what other operations might be able to be performed at wells.[[7]](#footnote-7)

Sensors

The above components suffice for applications which don’t involve decision making, but future DMF platforms will almost certainly going to want to incorporate *sensors* to allow programs that branch or loop based on what’s found. The most likely sensors would be downward facing cameras. Cameras with modest resolution could look over the board as a whole and detect color changes in drops. High-resolution cameras could focus on individual pads and perform operations like detecting and counting biological cells of a certain type.

It’s possible that other sorts of sensors could be added to certain pads to detect other physical properties of a drop. (Conductance? Presence of magnetic beads?)

Current software

There are currently two software development activities related to the prototype DMF platform.

The Hardware Abstraction Library (HAL)

Aaron Bowdle is developing/has developed the *Hardware Abstraction Library (HAL)*, which I currently know very little about, but I hope to learn a lot more about very shortly. From what I understand from Viktor, it accepts a sequence of descriptors, one per time step, that specify the coordinates of pads whose electrodes and electromagnets should be on and the temperatures for the various thermal regions. The HAL then takes this sequence and interfaces directly with the board to realize the specified sequence.[[8]](#footnote-8)

Wombat

Viktor Shkolnikov is developing *Wombat*, a Python-based library for higher-level DMF programming. Wombat-based programs are defined in terms of *super operations*, such as mixing, washing, and incubating volumes of reagents, and each super operation takes its input reagents from wells and stores its output reagents into (previously empty) wells. Given a sequence of super operations, Wombat determines the wells that the initial reagents need to be in, the output wells for each super operation, and the sequence of low-level steps that are needed to realize each super operation.

Wombat also includes a rudimentary graphical emulator that takes a generated (HAL-like?) state sequence and visualizes movement of drops on the board.

Thylacine

Next, I’ll describe the suite of languages and tools that I envision building. Note that this is essentially the plan that I came up with before I came on board and had access to any internal information. While I haven’t yet heard anything that would cause me to think that this is not the right approach, I am fully prepared to be convinced that it needs to change.

Assumptions and expectation setting

Before describing the languages and tools themselves, it’s probably worthwhile to lay out some of the assumptions that I’m making.

Target users

I have two primary users in mind in my designs. The first understands the domain and has some minimal programming experience, likely in a language like Python or maybe basic Java. They’re happy to be trained on a special-purpose language for this new platform, but they don’t want the code they write to be too closely tied to one particular brand or version of the platform. I expect this user to write *recipes* in MRL and run them through the MRL compiler to generate code to run on their particular hardware (specified by the MPDL specification provided by the hardware manufacturer).

The second user is a domain expert who has no interest in learning to program. This user wants to take published protocols and do the minimal translation into MPSL specifications, which they will be able to read and be confident in. They will then run the MPSL compiler to generate code for their platform. Alternatively, the published protocols themselves could be specified in MPSL, downloadable from the publication’s website or a publicly available database.

Interactive MPAM API

Some users will want to drive the board interactively. The MPAM API will be directly callable from at least Python, allowing interactive control of the board at a level higher than needing to worry about specific electrodes, but lower than platform-independent recipes.

Implementation languages

For the tools themselves, I’m assuming that I can choose the appropriate programming languages for each, and the code doesn’t need to be understandable to users who aren’t professional programmers. (It does, however, need to be maintainable.) My current expectation is that I will use a combination of C++ and Java and will use ANTLR to generate parsers.

Code quality

My current code quality target is what I’d call “high-quality propoty”. It will be as efficient and robust as I can make it, and it will be commented and documented such that others can (hopefully) understand it. As I won’t have a Q/A team beating on it, and because it will venture into some areas (e.g., routing algorithms), I’m not an expert in, it will probably not be product quality.

It will certainly be usable and releasable as an open-source reference implementation (if desired), but a larger team with different skill sets would be required to turn it into a revenue-generating product or, likely, even an open-source project that customers of the hardware platform could rely on for their own mission-critical applications.

Languages

There are four languages that I am planning on designing. Repeating

1. I’m tempted to call them “cells”, but that might be confusing for applications that involve detection or manipulation of biological cells. [↑](#footnote-ref-1)
2. This would also seem to happen if you turn on the electrode of a non-empty pad with a drop in a neighboring pad, but my understanding is that it’s considered a bad idea to have drops in neighboring pads because of possible interactions. [↑](#footnote-ref-2)
3. The current pad arrays are rectangular, but this constraint would seem to mean that it might be advantageous in the future to consider hexagonal pad tilings, as then each pad would have six neighbors and all of a pad’s neighbors would be across an edge rather than across a corner. [↑](#footnote-ref-3)
4. I’m really fuzzy on how this works. [↑](#footnote-ref-4)
5. Narrator: He doesn’t. He’s looking at a picture of an OpenDrop board and trying to reverse engineer what happens. [↑](#footnote-ref-5)
6. Which I will understand as soon as I ask somebody about them. [↑](#footnote-ref-6)
7. Just off the top of my head, it might be useful to be able to extract fluid from the well to off-board equipment to perform operations like centrifuging or agitation. Extraction would also be useful for applications in which a sample is prepared for later use. [↑](#footnote-ref-7)
8. It probably does a lot more than that. My apologies, Aaron. [↑](#footnote-ref-8)