Tidepool: A Modern and Open Data Architecture for Diabetes Management

Steve McCanne, Founder and CTO, Tidepool
steve@tidepool.org
June, 2013

Years ago, Marc Weiser, a renowned computer scientists at Xerox PARC, famously defined the notion of "calm technology" :

- *The purpose of a computer is to help you do something else.*
- *The best computer is a quiet, invisible servant.*
- *The more you can do by intuition the smarter you are; the computer should extend your unconscious.*
- *Technology should create calm.*

Introduction

Today's diabetes industry is built upon proprietary architectures, where each vendor produces a set of devices and software that is closed and interoperable only within their product line.

This model leads to several problems.

First, doctors and nurses must learn all the different systems that their patients collectively use. They must also look at multiple screens or multiple printouts of data from different devices to formulate diagnostic decisions. There is no integrated, holistic view of all of the data that comes from CGMs, meters, pumps, activity monitors, and so forth. This leads not only to inefficiency but also contributes to higher rates of human error.

Second, device vendors have, for the most part, failed to excel at software design and development. With rare exception, it is difficult for companies to develop core competencies that span the boundary between hardware and software. As a result, the software systems that clinics and individuals have at their disposal are poorly designed, are not easy to use, and can lead to suboptimal health outcomes. A huge reason why people don't upload their data and look at it is because it's hard to do and it's hard to understand. In short, it just doesn't really help that much, not because the data isn't useful, but rather because the software quality is lacking.

Finally, and perhaps the most troubling aspect of the proprietary architecture is its closed data model, which we believe leads to suboptimal market dynamics. In particular, the data that is generated by devices and stored in software systems is closed and there are no existing open APIs for accessing diabetes data. This means that if

someone or some company comes along and wants to build innovative systems that interface with existing devices, it is difficult.

An Open Data Architecture

An exciting and powerful model has recently emerged for building web-based applications. The widespread emergence of mobile devices has led to a new architecture where all apps — whether it's the browser running on a desktop on a large-screen display or native code running on a phone — intercommunicate with application infrastructure through network-based APIs. In this way, a desktop browser, a tablet app, and a phone app can all speak to a common infrastructure backend allowing the user to access the app anywhere and through any platform.

This emerging model has greatly accelerated the pace of innovation on the web. By leveraging open source and modern, agile design patterns, small teams are able to build meaningful and easy-to-use web-based systems in relatively short time frames. Developers speak of “coding by Google” because so many programming problems have already been solved and developers can do simple web searches to find open-source implementations of what they need. In other words, with modern software architectures, the rate of development has gone up significantly, and the costs have gone way down.

Our proposed Tidepool architecture follows these modern design patterns. In this approach, a single-page app runs as a client-side JavaScript program interacting with one or more backend services via RESTful APIs. Here, the app is embodied in a “blank” (or very minimal) page of HTML along with a JavaScript program that implements the user interface of the web-based application with programmed logic rather than HTML presentation from the server. This JavaScript client stores and loads application data and performs transformative functions via REST calls to services on the Internet, which may or may not be co-located with the web server that hosts the application (i.e., via CORS).

A key benefit of this approach is that nearly all of the application logic is embodied in the client rather than in a complex, multi-tier service architecture. The application state is maintained in the client and any long-term application state is persisted to the backend through simple, RESTful interfaces. Furthermore, because the back end consists of a set of well-defined REST-based APIs, different clients can be built that talk to this common backend allowing a clean separation of application design from back-end semantics.

A Distributed Set of Device Clouds

This modern approach to web application design is a great fit for diabetes devices and management systems. In our version of the future, the industry moves from proprietary data architectures to open data architectures. Vendors can still innovate and compete within the proprietary aspects of their devices and components but the data associated with the user would be ultimately owned by the user and accessible through open APIs.

There is no single entity that owns this “system architecture” just as there is no single entity that owns the design of the Internet.

In this open world, the term “cloud” is often used to refer to infrastructure that provides web-based services for storing data and performing computation. While some projects like RunKeeper Health Graph and the Human API envision a single cloud service that aggregates all the data from many different devices, we believe a more likely future will involve a distributed set of vendor clouds. In this approach, each device vendor develops their own set of software and exposes data from their device through open, REST-based APIs in line with how many of today's health and fitness devices are already deployed.

Since a centralized cloud architecture is a special case of a distributed cloud architecture (where there is only one cloud), it is sufficient to design for the latter case. In this fashion, either a central or a distributed model will be compatible with our architecture, so by this line of reasoning, our architecture is agnostic as to whether the centralized or distributed cloud wins out.

That said, we believe competitive dynamics and the “coopetition” aspects of this market environment will lead to an open but fragmented structure. This is already evidenced by the emergence of open APIs for fitness devices, social networks, and so forth, where the APIs remain open and developer-friendly yet the corporate entity behind the API still retains a certain amount of influence and control over their users and platform.

The Software Stack

If the distributed cloud architecture prevails and each vendor has their own device cloud with their own proprietary apps, how would it be possible to create new apps that integrate across all of these devices? Well, this is the beauty of the new, single-page app web architecture. As long as each cloud vendor supports an open API and a modern trust model (e.g., based on OAuth), then web-based (and native mobile) applications can be developed that reach out to multiple device clouds, pull data to the client app using AJAX, and present an integrated and holistic view to the user.

For example, an app could be built that uses Facebook to login and have an online conversation, and at the same time the app might use the RunKeeper Health Graph API to access fitness and activity data. For CGM data, it could then access the device vendor's cloud to pull down the latest data from the user's CGM. And similarly, the app could access the insulin pump vendor's cloud to access the latest pump information.

This approach, however, leads to an interoperability problem. If you build an app that speaks to all of these different services, you must understand the protocols and data formats of every vendor's service. One approach would be to try to get all the vendors to agree on a specific set of protocols and formats, e.g., through standardization efforts. While standards would be hugely helpful, getting everyone to agree is a lengthy if not impossible process. This just won't work.

Instead, we believe a better approach for solving the interoperability problem is through an open-source software “stack.” In this model, a set of software components are developed that speak to each of the vendor's clouds and translates from vendor-specific “language” to the common “language” of the software stack. For example, a module could be developed that understands the vendor-specific format of a particular insulin pump's data and translates this information to an internal format that is used by the rest of the stack. In particular, a module that implements the visualization of pump data can then consume the common format to render the visualization. This design theme would persist across the entire application stack. In this way, adding support for a new device type simply involves writing a module that translates the vendor-specific information to the common format.

Of course, we envision this stack doing more than providing a translation layer for vendor-specific device clouds. Over time, the Tidepool project and an engaged community of open-source developers and contributors will assemble a rich set of modules with a broad range of functionality like state-of-the-art visualizations of diabetes and activity data, interactions across social networks, various authentication methods (e.g., from social networks to clinic EHR systems like EPIC), the ability to experiment with algorithms for analysis and recommendation, and so forth.

A Bridge from Closed to Open

While social networks and fitness devices are all following the modern, open paradigm for data interchange, the existing vendors of diabetes devices are decidedly proprietary. This begs the question of how can we get started with the software stack if the data lies behind closed doors?

Our approach is to build a bridge. A bridge from closed to open. It would be ideal if we could build a software stack that seamlessly interoperated with the existing diabetes device vendors, but this is difficult given the proprietary nature of existing systems.

Instead, what we will do is make it as easy as possible on the user given the proprietary boundaries. For example, a user could simply drag and drop raw data reports by downloading data out of their proprietary app, e.g., to a CSV file, then simply dragging the CSV file onto a Tidepool app that integrates the data into an open (but still private) cloud data service compatible with the Tidepool software stack.

Another approach could be more automatic, where a user installs software to login into their vendor site and assists them by automatically transferring their data from the proprietary platform to their open platform.

The Tidepool project could also support a model where we develop software that speaks directly to certain devices in collaboration with the device vendor, who might agree with our vision and want to work with Tidepool to help move their business into this modern age.

If we can build an application or two that are compelling enough for people to want to use despite some of the integration challenges, then we will grow a user base which can lead to a critical mass. At some point, device vendors will recognize the business advantage of interoperating with the Tidepool platform. Once this happens, then more users will find compelling utility in Tidepool applications, the system will become self-reinforcing, and the transition to an open architecture will become inevitable.

blip: A Foundational Application

A ubiquitous software stack and open standards would solve a lot of problems. However, designing and building such a stack is difficult without the context of an actual application. And, building an application by brute force without any concern about reusable architecture won't have the impact we envision.

To this end, we are developing an app called "**blip**" as a vehicle to exercise and inform the design of the Tidepool stack. **blip** is a single-page application that runs in a web browser, provides easy-to-use yet powerful visualizations of diabetes and activity data, and anchors an on-line conversation using social networking so the PWD, their family, and their care providers can all interact online with low effort and high efficacy, capturing the "learning moments" of diabetes management when and as they occur.

Enter JavaScript

One of the challenges of building a single-page application is the inherently asynchronous constraints of the programming model. Importantly, the user interface must remain interactive and "snappy" while data is being loaded and manipulated across the Internet. Moreover, this communication involves data transfers and updates to multiple device clouds located in different parts of the Internet, typically using REST APIs and data encoded in the popular JSON data format. To do all this, the client-side application must be able to send multiple REST requests to different places in the network, manage the responses as they return, update UI elements in response, and keep the user interface active all while this is going on. In other words, the program must handle a potentially large number events in an asynchronous fashion and manage the complexity of different programming actions completing in arbitrary orders.

Fortunately, JavaScript includes several key language features – e.g., closures, anonymous functions, and object literals – that come together in a powerful way to make an "evented" style of programming highly effective. From these language features, a number of powerful design patterns have been developed over recent years and packaged into re-usable libraries.

A large number of such libraries have been released as open source to facilitate the development of single-page apps in client-side JavaScript programs. These libraries tend to (loosely) follow the traditional model-view-controller (MVC) paradigm of user-

interface design in one way or another (though there is a bit of a background controversy over the precise definition of MVC and whether the different frameworks libraries actually conform to it).

Examples of such libraries, usually called an “MVC framework” or just “framework,” include backbone, ember, YUI, ExtJs, angular, knockout, and so forth. Each framework has its strengths and weaknesses and none of them quite cover the same range of functionality. Moreover, none of them has emerged as the clear winner. In short, this space is all still a bit of a moving target and is highly fragmented, though we expect things to shake out somewhat in the coming years.

That said, we favor the model championed by backbone. As opposed to a batteries-included, we-do-everything-for-you approach, backbone is a relatively small library that focuses on the core functionality of the MVC pattern. The intent is that the programmer would pick and choose other libraries implementing their preferred style and functionality to build up a more customized development environment. This approach is often called the “micro framework” approach since it involves composing many small (micro) frameworks together in a custom arrangement.

We believe this micro-framework approach is a good fit for **blip** and the Tidepool architecture as we intend to provide a broad set of functionality for the diabetes ecosystem, where different Tidepool modules can be mixed and matched and plugged into other environments in an agile fashion.

To this end, our directional thinking revolves around an architectural model consisting of:

- a (JSON) data schema for representing diabetes related data sets
- a set of REST APIs for our backend
- a client-side, JavaScript-based software stack for desktop and
- mobile app development based on modern, open-source libraries like
- backbone, d3, jQuery, and so forth
- an infrastructure-side stack for app data persistence and analytics

blip Functionality

Given this architectural model and software stack, we now turn to the functionality of the **blip** application itself.

To make things easy, **blip** must do much of the work automatically, behind the scenes, for the PWD. The first step is collecting all of the relevant data in real-time and in one place: carbohydrate consumption and insulin dosage from pumps; blood glucose levels from finger pokes and from continuous glucose monitors; and even activity data from devices like FitBit and Nike Fuel Band.

This is where the device clouds come in, as well as the closed-to-open bridges, described above. By first laying the groundwork in the Tidepool stack, the **blip** app can

be written using the stack. This resolves all the issues with data collection, aggregation, and transport have been resolved. In other words, the developer can simply invoke the functionality of the stack without having to write the code to manage the complexity of all of the diabetes data.

Another key consideration is that **blip** must be easy. It must ask people to do less not more. The key here is most all of the burdensome data collection and integration is completely seamless and hidden behind the scenes. Ideally, all of the data required by the **blip** platform would just "show up" in the system. In the near term, however, the PWD might have to deal with bridging data from the proprietary world to the open world. We will make this as easy as possible, until at some point in the future when the PWD should be hardly involved at all. This will happen when the vendors converge their device data generation with on-line clouds.

We also have a future vision of the Tidepool platform providing automatic analysis and recommendation (though we realize the challenges of providing actionable advice with respect to FDA regulations and so forth). The idea is the underlying data can be analyzed and cues automatically generated from expert intelligence that is built into **blip** and helps the PWD. This intelligence isn't magically all programmed into **blip** by engineers from the start. Rather, it is assembled over time through interactions between endocrinologists who, after many years of practice, have obtained incredible intuitions and pattern-matching abilities around the tremendous variability of the diabetes feedback-control system. The **blip** system will include an interface for adding expert analysis modules so that the feedback-control system can be continuously improved. A doctor selects and configures a set of modules that is relevant for each PWD under his/her care.

A consumer-mediated model using Backend as a Service (BaaS)

To make all this work, applications like **blip** must be hosted and deployed somewhere. Our client-side JavaScript model makes this straightforward. By leveraging existing cloud services, in particular, the recently emergent "Backend as a Service" (BaaS) providers, the amount of infrastructure that must be deployed is minimized.

Unlike traditional web architectures, where the server infrastructure implements most all of the application logic, in our model the client-side JavaScript running in the browser implements most of the logic. This means to host such a web, we simply need to statically serve the JavaScript code and any static elements like CSS style sheets and image files. These elements are all cacheable and require a relatively small amount of server capacity since the heavy lifting is done on the client.

Once the client JavaScript code has been served and loaded into the browser, the client reaches out to the various devices and other clouds to fetch the user's data and integrate it into the client-side. For example, the client-side JavaScript could reach out to the parse.com BaaS for core configuration data, then mashup activity data from

RunKeeper, pump data from an a vendor cloud, and CGM data from a second vendor's cloud.

In one approach, rather than have Tidepool proactively fetch data on behalf of the user, the PWD could sign up with a cloud provider that provides the user a “data vault” and Tidepool could merely supply the software that would allow the PWD to sync their various diabetes data into their data vault. In this way, the *user* owns and orchestrates the movement of *their* data between clouds, not the provider of the application.

As another example, a new vendor (or even Tidepool itself, TBD) could assemble a clinical software system out of Tidepool system components, perhaps adding their own proprietary elements, obtain FDA approval, and offer a patient-first model for clinical care. Here, the PWD could choose when and how to interact with their clinic.

The Challenges of Regulatory Compliance

This brave new approach is awesome. Yet, the existing regulatory laws and agencies have yet to understand or embrace these new models.

We believe in the FDA and laws like HIPAA. They are important and valuable to protect patients' interests and rights. We will work with these agencies and conform with all legal requirements.

That said, we also believe we should invent the future we desire, and not be trapped by the constraints of how agencies like the FDA work today. We will partner with the FDA and figure out what we collectively want. There is absolutely no reason why we can't have a world where a range of FDA-approved cloud-based data providers interoperate with FDA-approved devices from a range of vendors, with third-party FDA-approved apps that ride on top of this platform, and it's all “safe and effective.”

It will take effort but we can get there. And, we can start by conducting safe, clinical research trials to prove the efficacy of the models before embarking on efforts to scale them and make them mainstream.

Conclusion

Clearly, getting data from devices into the cloud is the big, hairy problem. Not technically, but “politically.” Yet, at the end of the day, all of this data is the PWD's data. The device manufacturers have to provide it to the patient and their doctor. Unfortunately, they don’t have to make the process of obtaining your data easy, especially if we want to automate the data capture.

Herein lies our opportunity. If Tidepool and **blip** could some how make it competitively advantageous for a device vendor to integrate with cloud through a set of open device clouds, then everything falls into place.

How do we make this happen? We need leverage. And leverage can come with a following. Bottoms up. A social uprising, so to speak. We will build an online community that has mass. We will build a web application like **blip** that is cool and easy and makes life better, even if the data collection is suboptimal. We will get PWD's and parents excited about this completely new approach to diabetes management and build enough the system to make them realize this is hugely transformational.

We build enough of the system to prove it out. To get a following. To draw people in. Maybe it's not perfect and there is an extra step or two. But people get it. And they see how the extra step or two could be removed if the device vendors had a clue. And they demand it. And the first device vendor who is Tidepool-compatible wins the race and takes market share and the rest are left scrambling to catch up.

It's a better world for so many people.

It is exciting.

Let's make it happen.