OpenApp

OpenApp is building a suite of modular open source tools to support transparent, democratic and decentralised organising. These tools seamlessly connect you with your community and other organizations around the world. [oriented to the global commons]

[OpenApp is a technical bridge between separate (social / cultural) verticals?]

..start your own virtual village, decision-making, collaborative budgeting, skill-sharing, timebanking, a local virtual marketplace, are all on the roadmap, or ready for alignment.

...
The web is intended as a set of **open common protocols** - common protocols allow anyone to publish and link to other websites, no one owns the protocols, and anyone[1] can participate in creating or redefining protocols. While this simple, democratic idea still underlines the web, much of our experience of the web now occurs on **closed mega-scale platforms** built over the original idea**.**

What should be simple connections between **people and groups**: friends and family, buyers, sellers, and collaborators must pass through global platform intermediary. Facebook for social networking, Ebay for trading, and Paypal for transacting etc. Connecting over these platforms means you have to pay their fees, view their advertising, and submit to their terms and conditions. [It's like living in a shopping mall](link to slum blog post, http://idlewords.com/bt14.htm#decentralize).

[But it doesn't have to be this way. Some of the original web visionaries like Tim Berners-Lee and have been collaborating on protocols that gives the power back to the people. These protocols are now ready for implementation and…
[We should be able to connect on our own terms, without having to go "off-the-grid".]

[1] as long as you speak English, and are willing to participate in drawn out email threads.

The Age of Platforms

Most networking and ‘sharing economy’ apps are **platforms**. The user signs up and joins a centralised platform from a single provider and connects to others on the same platform.

We view platforms as isolated silos that push entrepreneurs to compete rather than collaborate, waste effort writing the same boilerplate code rather than innovate, follow a risky strategy that ultimately degrades user experience and rights, and benefits financial investors over other stakeholders.

Building a platform obliges entrepreneurs to follow a ‘speed-to-market’ strategy - become the dominant player with the largest user-base as fast as possible before a competitor gets there first. The more users, the more people there are with whom to connect, trade and share, the more attractive the platform appears to further users. Platform builders dream of runaway ‘viral growth’ when their platform gains enough gravity to pull in users with little direct marketing effort.

Unfortunately, growing rapidly is expensive and providers can’t initially charge for the service lest they scare users away. It almost always involves large amounts of investor cash who will want a financial return at a later date and stiff competition from rivals offering similar services.

If a platform survives and becomes the dominant player an uneasy relationship forms with the users. To keep investors happy the provider must now monetise by upping fees, or selling ads and user-data. Monetisation degrades the user’s experience and risks an exodus to a cheaper, shinier competitor. Users might get a useful service but give up rights to their data and control of their online experience.

At the same time, it can be difficult for different platforms working on similar ideas to collaborate, **even if the platforms are open-source**. If someone starts a similar platform you might worry that they will take all the users.

If someone makes an app that uses the same protocols then they've just **added** to the ecosystem, and made **your app** more valuable.

The [summary]..

The Protocol Renaissance

OpenApp is following a **common protocol** based strategy that turns platform logic on its head. We want users to have the power that comes from accessing a large network, but we don't need, or want, to *own* the network. Instead of creating silo'd proprietary networks and piecemeal integrations,

This way, competitors become collaborators. When everyone participates

In a protocol We don't need to agree on everything, just where we intersect.

It's not the wires, its the computers
Its not the computers its the documents
Its not the documents, its the things
http://www.w3.org/DesignIssues/Abstractions.html

We want to go beyond linking apps as a service, we want to provide the infrastructure so that the default of any new app is to be linked to all other apps.

"Managed Trust"

Projects like Ethereum, Maidsafe and OpenBazaar have similar goals. We think these projects are cool and follow them with interest. The difference is that these projects assume users need to strong cryptographic and anonymity protection, and interact with “zero-trust”. We can see the value of doing this in many use-cases, but rather than attempt a “zero-trust” distributed system, we aim for a “managed-trust” local system. “Managed-trust”, means

. By local, we mean we “each instance of OpenApp” emerges from your local identity, as a person and as a member of groups. (TODO: look up how Diaspora describes federated systems)

(modular vs monolithic)

Open Source at scale: http://youtu.be/Vm3_F9Ln9lo
http://substack.net/many_things
http://blog.izs.me/post/48281998870/unix-philosophy-and-node-js

allow developers to focus on their contribution to the ecosystem.

OpenApp standards (*work in progress*)

- vocab ontologies: OWL
- type schemas: JSON-Schema v4
- services:
    - content:
        - JSON-LD
            - JSON services can have translation layer to become JSON-LD services
        - Triple Pattern Fragments
    - interfaces:
        - HTTP
        - WebSockets
        - libchan
    - serialization formats:
        - JSON
        - msgpack v5
        - Transit?
    - hypermedia formats:
        - JSON-API?
        - Hydra?

- user interfaces: Web Components
    - while Web Components is still unsupported in browsers (and suffers from bad performance), we may temporarily standardize on something like React or mercury, which can then be Web Component-ified later.

OpenApp.js stack:

- permanently solve inter-app linking
    - no more pairwise app integrations
- simple:
    - model: immutable Maps, Arrays, Sets described by JSON-schema
    - view: Web Components
    - controller: set of functions exported over an interface as a micro-service
- truly modular architecture (*work in progress*)
    - type schemas:
        - allow JSON-schema v4 to include vocab metadata, which can be used to export JSON-LD contexts
    - micro-services:
        - method: function with input type and output type
        - trait: Class instance mixin
        - hook: before/after function on methods
        - service: Class with data, methods, traits, and hooks that emits events on method calls and call methods on event emit
        - store: service with find/get/create/update/remove methods
        - transactor: service to request data writes
        - index: service to subscribe to data reads
    - user interfaces:
        - compose web components from service type
    - service containers: Docker (using libcontainer)
    - service orchestration: libswarm
- creating modules is simple
    - types
        - describe a linked data type with a json schema
        - get a real-time, linked data service for free!
        - get a reactive interface for free!
    - skins
        - write interface styling
        - hot-reload skin updates
- compose modules into apps
- developing for stack should be a live experience
- interfaces should encourage raw views of state

first release: ‘the innkeeper’

- identity (people and groups)

- simple authentication with Mozilla Persona
- simple interfaces
  - describe entities with JSON-Schema
    - Linked Data (triple store) storage is automatic
    - CRUD REST/WebSocket services are automatic
    - CRUD REST/WebSocket clients are automatic
  - create React components for entities
    - contextual CRUD pages are automatic

further releases:

- marketplace (offers, requests, transactions, receipts)
- federated data queries with Triple Pattern Fragment
- context-based store traits (data, type, function)
- data, context, interaction architecture
- transactional data (persist immutably)

---

notes:

new renaissance, semantic web

dark ages
[1st renaissance]
age of empires
2nd renaissance

- Dark ages
  - intra-net platforms
  - local cathedral
- [1st renaisaance]
  - inter-net protocols
  - local / global bazaar
- Age of empires
  - intra-app platforms
  - global cathedral
  - http://www.forbes.com/sites/venkateshrao/2011/10/10/public-computing-and-the-next-gang-of-four/
- [2nd renaissance]
  - inter-app protocols
  - localized to identity (as person and part of group(s))
  - globalized in semantic web
  - local / global bazaar

The vast majority of apps built today do not use this architectural style or meet these standards. As a result, these applications often end up as difficult-to-maintain, isolated silos.

We're trying to shift this paradigm. We believe a democratic web is built on shared *protocols* not *platforms.*

- *Protocols over Platforms.*
- *Islands vs Ecosystems*
- *Centralised, Federated, Decentralised.*
- *Opensource*
- *The "Open Graph"*

Protocols over Platforms

Facebook, Google, Amazon let people to connect, share information, and trade. Unfortunately, this all happens on proprietary *platforms*. Platform owners control the user's data

No-one wants

By taking advantage of the latest linked data standards OpenApp apps can connect to multiple apps, across multiple servers. No matter where the Users will no longer be tied to particular platforms.

Serving documents to serving data

Through the 90's a website server (the computer that holds the application code) would serve an html *document* (a web page) and web browser would display that *document*. That was basically it. The user might submit a form and the the server would create a different document and reload the webpage. The user could click on a link and navigate to another document (perhaps served from a different computer to the first). For the user, navigating between different html documents was *seamless.*
Or was it?

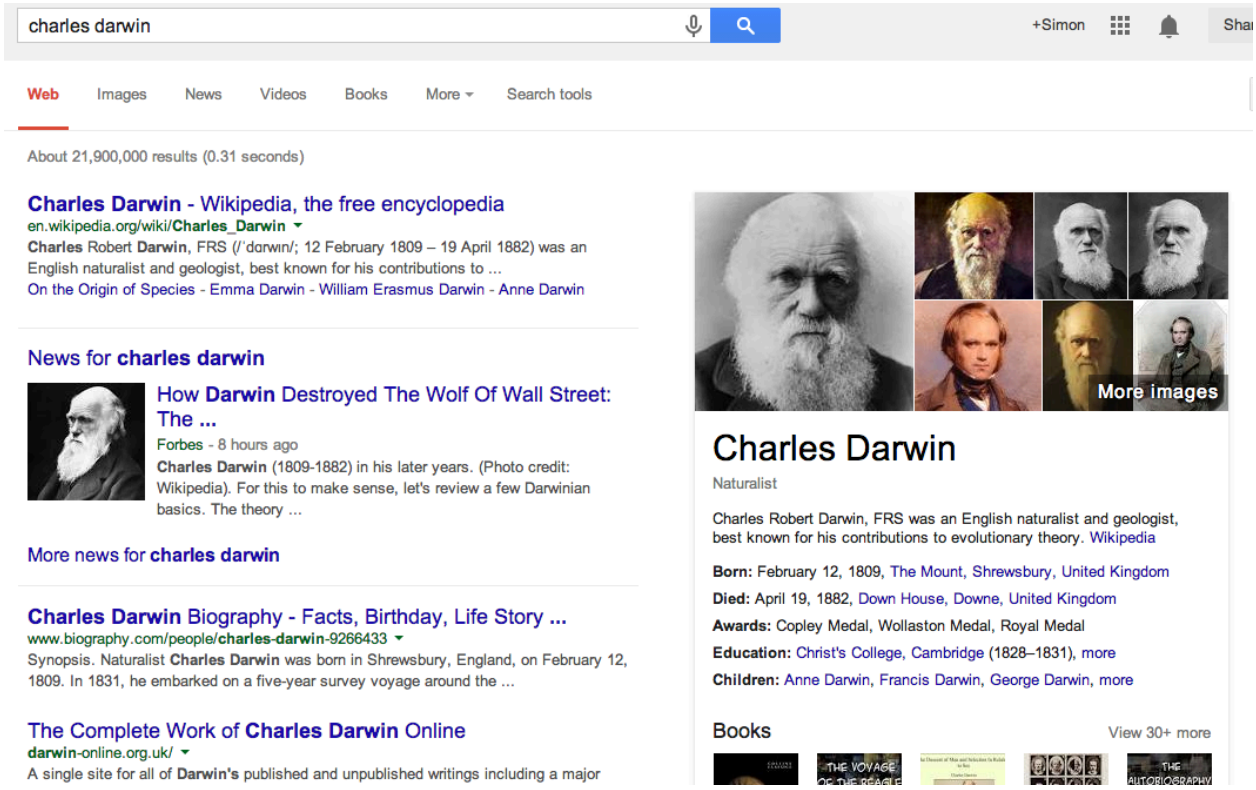
Beginning around 2000, servers also began serving *data*. Data arrives in the user's web browser, the browser and the application code then uses these data to create and display a

document. The shift in web technologies allows features like ‘live updating’ -the web page can display new content without reloading.

Applications could also pull *data* from *other applications* and display it as they liked within their own *documents* -a stream of tweets or a google map could appear in a *document*. Users didn’t have to follow a link to see relevant information. For the user, viewing data within a web page was *integrated* and *seamless.*
Or was it?

Linked Data is the next evolution of the web. The Linked Data user proposition is that “users are interested in *things* and the *connections between things* more than they’re interested in *documents* about things”* Have you noticed recently how if you google for a specific thing (a person, place etc), google will show you a profile of that thing to the right of the page. When you search for “Ruby”, you want the the search to narrow quickly to “Ruby the programming language”, or “your friend named Ruby”. We want to give users a similar rich search experience through the Open-App UI and prioritise information that the user cares about.

The Linked Data social proposition is that “by structuring data in common vocabularies and formats we can enrich a massive, growing, common, decentralised database” Apps that are interoperable with this database grow the data commons and benefit as the database grows. w



*Sometimes you are searching for a specific document, but get this, a document is also a *thing. [mind blown]*