Problems present in organizing

I will start this text by pointing out problems that exist with the (horizontal, or in other words grassroots) organizations of our resistance movements and a potential solution that I've been working on that I am also asking for help with.

These problems are going to be mostly pertaining to how information is managed in horizontal groups, since that is what software is most able to address. The general problems I lay out here are:

1. Not giving enough clear information to potential volunteers and members for them to form accurate expectations.
2. Cognitive overhead in using multiple platforms for the various functions of complex organizations like catalyst orgs; project management, communication, and decision making must be done in different platforms that can't communicate well
3. High menial labor in performing both regular and infrequent administrative tasks, also with a high risk of error.
4. Lack of data protection (e.g. during a house raid) in the ad-hoc practices of horizontal orgs, putting members in danger.
5. Lack of customizable ergonomics in software interface
6. Lack of control over software changes; software that the org uses are created by corporations or capitalist entities in general, who may enforce software updates that sabotage the functioning of the organization
7. Software's generality doesn't fit specific org processes

These problems can make organizations less ergonomic and tiring (perhaps even dangerous) to participate in, so solving these problems could be important for ensuring the longevity of organizations supporting radical movements.

How and what kind of software could help

So, then what could be done to solve these problems? Of course, there can be changes that the organization itself can make to solve the problem without any need for software. Indeed, face-to-face communication is a better choice in various situations, but insofar as the organization needs to maintain information that is suitable for above-ground operations, electronic communications can be convenient especially when schedule conflicts or other events disruptive for in-person interactions occur.

So, if we suppose that an organization did have its own custom-made software for its internal and external functions, members would be able to collaborate together on their own tasks, while still being able to communicate across the general council of the organization for important decisions and meetings. The network could be built upon a custom network (e.g. See this blog post by coop Denton) to keep things secure from external entities, and individual members would be less stressed by tedious and unergonomic tasks, making them more able to carry out more tasks to build power.

If we look at the spectrum of software that will help organizations then, something like an assortment of excel templates & macros or a very specific high-level app could help for a specific group or type of group (eg. The app that Bike Brigade made for their food distro project) could help in many cases.

However, another option could be a form of framework software that could have a broader appeal across many types of horizontal groups. Since horizontal groups are going to vary so much across localities, giving people the tools to build software according to their conditions could be very beneficial.

There has already been various software out there that has carried this out to some extent; the two I'm most aware of that is most relevant to this text is Dual Power App, and Karrot.

Unfortunately, BSA, the org carrying that out, seems to be hibernating as of now and the app is not finished. I took much from the notes and discussions that formed the ideas and architecture of the app, however I am going in a bit of a different direction than them which I will elaborate on later on in the implementation notes.

As for Karrot (which is used primarily in Europe to my understanding), it is close in that it's for horizontal orgs' general activities as a whole, but to my knowledge doesn't seem to have horizontal decision-making processes, project management, a focus on security, or the ability to be p2p.

This is where I get to the main aspects of what I am attempting to build and why I think horizontal organizational software should be designed in such a way:

1. Distributed peer-to-peer network – instead of requiring to host all data on a central server, I will make the application store it's data on everyone's device, and communicate within that group of everyone's devices. This makes the network more resilient to repression as everyone's devices would have to be seized or hacked instead of one, and if such a thing were to happen, the data could be destroyed for that particular device without much loss for the organization (assuming the victim had enough time to do so).

2. Framework software – Instead of making an opinionated application with only some measure of customization, I am planning on making this application framework software; or to be more precise, a sort of engine application whose entire purpose is to build the top-layer part of what the application should look like for the organization; like a game engine in game development. The software would just provide the tools for people to create such an app in an editor interface. There could also perhaps be a few ready-made applications for a specific group type after the core engine is made, e.g. a symbiosis cooperation group.
3. Focus on security – I plan for there to be two main aspects to the security of the application: the active security and the passive security. The active security will be a specific component within the application that actively watches for malicious activity and log/notify users of it, or even take automated actions. The passive security aspect will be comprise the total design of the architecture and code; the book "Secure By Design" inspired most of what that comprises (it's a lot) but in basic terms of what I've taken from the book, I have based the application off of a domain model which fits the target real-life environment of the software, and followed certain principles of both the code itself and the architecture to minimize the impact of certain threat actors' potential actions against the software. Of course, the more dominant of the possible threat actors like governmental agencies will be effectively impossible to not expose info to when devices are connected over the internet, but attacks by smaller threats like individual infiltrators and smaller groups could be addressed by these two approaches, which can decrease the likelihood of domain-specific vulnerabilities appearing.

Developer summary of prefengine

PrefigurativeEngine (PE) is engine software that constructs, and runs business software for horizontal organizations engaged in constructive and deconstructive activism. I speak of "business software" in the more general sense of pre-defined organizational functions being done by code, not specifically the needs of capitalist entities. By horizontal organizations I mean groups that make decisions via consensus or some form of direct democracy and that fight for social movements. Examples include catalyst organizations, mutual-aid groups, prison support, etc. PE does this by managing information regarding collective decisions, projects, and communication in general.

The above is achieved with a private network that maintains a distributed database, that favors consistency for it's stored data. It is capable of maintaining both peer-to-peer and server-like connections. Higher-level systems are built on top for humans to enter in information to and read information from the database and other peers.

Elaboration on specifics of prefengine

I will note here a few possible aspects of implementing this that I don't plan to focus on for this software.

I'm not intending for the software to function like a public space; what I mean is software like social media (e.g. Mastodon) and things like messaging apps where you can send a message to anyone. Instead, this will be private P2P (groupware to be exact) software intended for groups where members are expected to have responsibility for certain group tasks. However support for federation could enable prefengine groups to communicate with other prefengine groups.

Another note is that real-time functions, while something that could be added in at some point, aren't the first priority; things like video calls in Jitsi or document collaboration software like Google Docs. Instead for this application I plan to favor a more asynchronous approach where immediate communication isn't necessarily expected; for example, things like loomio's decision-making threads.

The last note here is that the org domain layer in particular will only be designed for horizontal groups wherein decisions are based on direct democracy/consensus.

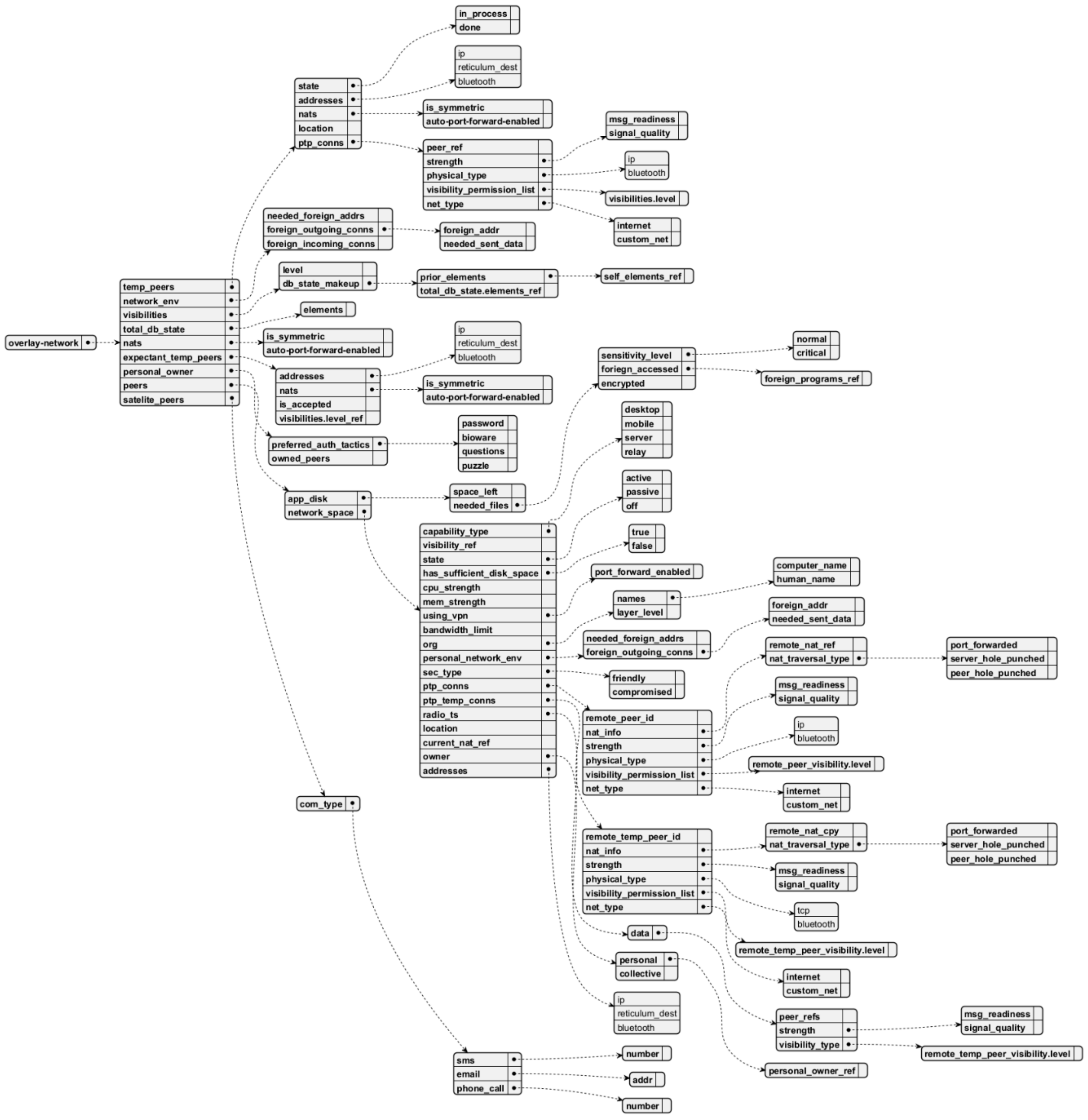
Nonetheless, some large components are planned to be modular in order to help others create other kinds of collab software; I've split up the application architecture into three large components: the technical layer (mostly involving the p2p functionality), business layer (involving functionality for chat, consensus, projects, etc.) and finally the api layer for a browser to interact with the software.

Another thing I wanted to lay out here is the domain models I've built, since I have taken some of the approach of DDD for this application (mostly because of how it's applied in Secure By Design). Although DDD could be said to put unnecessary limits on the code and require a bunch of unneeded work, in this case it is essential in how I am planning to make this program always do what is expected even in the face of attacks. Also, at least in my opinion, it will also be easier to reason about and make changes to down the line with this approach.

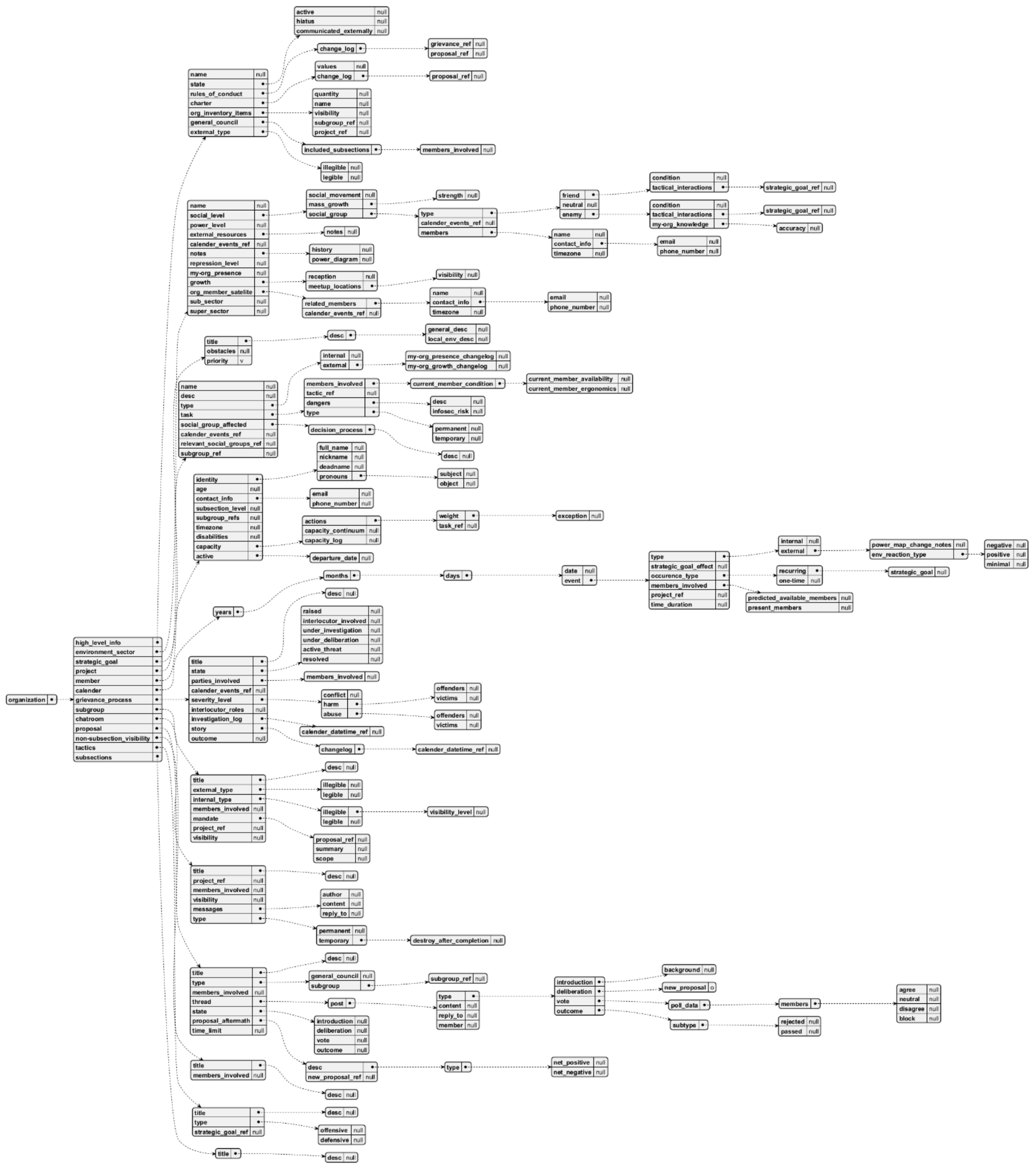
These models describe what situation this application is being built for, essentially the real-life organizational environment the software is intended to support. This way, it is clear what exact environment the application is made for, and therefore how it should behave when handling external input. I will talk about this more in the implementation section, but this model also describes how data should be modeled inside the code itself. I have made the technical layer model, describing the intended problem domain of that system, and the business layer model, doing the same for that system.

The technical layer model describes horizontal groups by the devices they use as well as other devices on the internet that those devices must interface with. The business layer does the same but on a human level.

Technical Layer model Illustration:



Business layer model (many entities are intended to be optional in a real org application):



I’ve built the above models to be relatively universal to different horizontal groups while still keeping a level of rigor to make the behavior more robust.

Potential features

Now that the application has been loosely defined and the general target of what this application is for has been defined, I will give several interesting features of what this engine could provide to coders and even normal people in order to provide effective business software to horizontal organizations:

ERP Editor:

The engine should include a web editor that can allow for the creation of web pages the api layer can serve later on, with dynamic content defined in user defined modules made with scripts.

Scripting:

Users should be able to create scripts with a high-level language (perhaps with Typescript) that have access to the suite of domain foundation objects, in order to create specific functions for an organization (e.g. a custom notification system). Users should also be able to load in libraries that may be disallowed from running if they are known to have critical vulnerabilities.

Physical Layer Abstraction:

The engine should also be able to perform all it's networking not only over the internet, but also over other physical layers like the LoRa protocol, and even a USB drive.

Application architecture to carry the above out

Here in this diagram, I have planned out what the subsystems in the technical layer looks like, which includes every box below the "Business Subsystems" layer. The various subsystem functions don't have to be custom implemented from scratch; they could be implemented by some library or other app. The placement of some boxes is a bit arbitrary, though I tried to loosely tailor the height level to correlate to how high-level some component is. This is also still a WIP – I haven't laid out specific functionalities in the business, api (defined as the HTTP Server and HTTP Client boxes), or frontend layers yet layers yet.

Editor Frontend TODO			
HTTP Server TODO		**HTTP Client** TODO	
Business Subsystems Organization Foundation | Data Mutation Rules | TODO			
Middleware RPC Message Lifecycle | PTP Messaging | Broadcast Messaging | Message Checksumming | Message Sequencing |	**Replication Management** Syncing Algorithm | Atomic commit | Visibility Management | Peer R/W Consistency	**Overlay Federation** Foreign Overlay Api | Federation Gateways | Federation Api	**Media** Video Compression & Transcoding | File Forensics Management | QR Code Encoder | PDF Renderer
Low-Level Server Physical Layer Interface | Anonymity Proxy (I2P | etc.) | Listener Messaging | Client Messaging | Encryption & Decryption | Packet Construction & Deconstruction |	**DBMS** SQL Adapter (Sqlite, etc.) | Object Relational Mapper | Caching	**Active Security** SIEM | Alert Events | Data Protection Commands	**Files** Local & Cloud File Abstraction | File Forensics Management | File Compression & Encryption

NAT Traversal (UPnP, etc.)			
Core Systems Logging | Event Manager | Localization | Engine Config | Cryptographic Primitives | Threading (Jobs, etc.)			
Platform Independence Platform Detection **|** File System **|** Monotonic Clock **|** Desktop Interface **|** Process Management **|** LoRa Hardware			

Implementation details

Finally, here, I want to lay out what decisions I have made in actually implementing all of this, and why I decided to make it in this way. This will only go over the technical layer as I'm currently working on it.

Probably the most important one is the language; I've chosen to pick from compiled languages like C or C++ for the technical layer, mostly for resource efficiency and speed.

Resource efficiency is important here because it could better enable people with low-end (e.g. low RAM) devices to use this app smoothly, even if the internal database starts to get large and the overlay network gets high in traffic. I also wanted to easily make this code run on embedded devices like LoRa transceivers and the like (which are being used by Coop Denton to make custom networks), which have strict memory and power limitations. Languages like Python still work on those devices of course, but it would take more memory and battery power to do the same operations a compiled lang could do.

Speed is perhaps a less important reason, but during high traffic moments I think it will eventually become a factor. For example, anonymity tools like Tor and I2P add a lot of load to message processing, and after that's done, prefengine itself will need to do a lot of processing at many points as well with how large of an application I plan for it to become.

This also becomes more important once networking for federations of groups (I mean prefengine federating here specifically), get implemented. That would enable scaling to many devices sending many messages.

I specifically chose rust over other LL languages based on a few principles I thought of:

1. Somewhat easy to make cross-platform
2. Easy auxiliary tasks like testing, docs, deps, etc.
3. Relatively popular
4. Memory reliability at runtime
5. Has libraries around p2p networking, business logic, and http
6. Object oriented-ish

The above principles I believe will facilitate quicker development and make it more reliable in the long-term.

I would definitely be more open to using higher level languages for higher level systems like the business and web layer however, since at that level a general-purpose language might suffice.

As for libraries, the most major library I've used at this point is reticulum, which I plan to use to provide the basic P2P networking for this application. It has essentially all the features that will be needed by the application except for the private group functionality.

More importantly though, it is a network protocol in and of itself, so it has good physical layer abstraction. Thus, the network will be able to work in many different contexts a group finds themselves in.

Pretty much the only big problem is that it's written in python; for now, I've made a basic python script for rust to communicate with, but in the future, I plan to try to port reticulum to rust at some point. There is already a rust crate that has the basic functions of reticulum implemented, so that could be used as well.