Method of Composable Business Process Automation via Email Protocol

Abstract

Email is the largest decentralized network on the web today.

- Decentralized, self-hosted and federated
- Used by 4 Billion Monthly Active Users
- Over 300 Billion transactions per day
- Used by 99% of consumers daily
- Used 2.5 hours per day
- 126 received per user per day average
- 75% of marketers use it with some automation

Email inboxes can be thought of as servers that are part of a federated network.
An email server can either receive emails sent by other servers, using the email protocol, or compose emails using the same protocol.
Email addresses consist of a domain (ex: username@domain.com (mailto:username@domain.com)), where multiple users can reside, and different access control layers can be configured: who may create an address, what permissions these addresses have.
Email addresses also consist of a user name (ex: username@domain.com (mailto:username@domain.com)), which allows sending emails to a particular recipient.
Lastly, email messages can be thought of as data payloads, that consist of information such as sender, recipient, text or HTML, as well as attachments.

Leveraging these properties of email servers, domains, addresses and messages one can develop a federated network of applications which serves as a viable alternative to blockchain-based systems, especially as it pertains to smart contract functionality. Email addresses fulfill the role of wallets and email messages fulfill the role of data payloads. Email inboxes serve the role of the distributed state layer and email applications serve the role of smart contracts.

"Best of both worlds"

Federated model made up of a network of email applications get the many of the benefits of a decentralized nextwork without placing unecessary burdens on the user.

It also gets the scalability of web2.0 applications with serverless architecture without compromising the composability.

Crypto adjacent. It's unbundling the innovative patterns from the hard to use crypto technologies.

It can appeal to a traditional investor if we can construct a traditional business model (TBD). Probably should avoid using a tokenized model. They are mostly appealing to crypto investors, and mostly on ideological grounds. The value proposition is strong enough to support a traditional revenue stream.

VCs are probably skeptical of the UX, complexity, speed or business models of crypto, but are probably intrigued and excited about the potential patterns unlocked and identified by the crypto space. They just see those are usable and implementable. We can achieve those patterns in a way that works for you.

These are characteristics we can unbundle from crypto:

- decentralization / federation aspect
- multiscale identity management (both individuals and groups)
- composable process automation (scalable beyond one hop: not just my or your process automation. Once you can use chains of certificates, you can trust the provenance of the data)
  We can introduce these benefits of decentralization, without the downsides of blockchain applications.
  We maintain the same benefits and remove frictions and costs.

Potential for an open core model?
We provide the core innovation, but monetize around the difficulties of the implementation, where people are not willing to put their time an energy (devops, overhead, setup, custom development).

Email as Decentralized Identity

In web 2.0, email addresses are the dominant form of identity. Log-in with Gmail is a dominant access point for services, and even when not integrated with email service providers the email address is used to identify a person or entity. Furthermore, group membership is denominated by email domain – specifically membership in a firm, academic institution, government department, etc is confirmed via having an email address in this particular domian. In practice, this leads to fragmentation where a person must have multiple email addresses in order to

interact with the digital world. Aggregation apps can help, but there is still a proliferation of unnecessary email boxes. Creating email accounts for group level identities (eg team email boxes) is basic step towards eliminating unnecessary attenion overhead.

++ Paragraph on decentralization

Email Applications for Business Process Automation

Replace/Rewrite:
Automations to organize, triage and even respond to emails according 'recipes' are unlocked. Such automations can save time at the individual level by eliminating rote tasks. Furthermore, automations properly applied allow the user to reclaim agency over their digital life. Today even the most senior executive is "below the API"— a slave to her inbox which is effectively a todo list forced upon her by others. With email applications she can retake control of her attention allocaion by defining her own channels, content aggregations, notifications and more.

Email App Architecture

Email Apps have the following properties:

Core Logic

- an *email address*
- takes inputs from external sources including but not necesarily limited to emails arriving at the App's *email address*
- has methods which may be triggered by the arrival of those inputs
- those methods may update the internal state of the email App
- those methods may emit events which are messages broadcast out to other applications. Message formats include but are not limited to sending emails from the *email address*

Rights/Access Control

- mappings from external accounts (other email addresses) to roles with the Email Application (uniquely identified by its *email address*)
- roles are comprised of packages of rights
- rights give access to specific methods (eg, emails from an email address with a role can trigger restricted methods whereas other emails may not)

- a method is a program that executes within the context of the email app as defined in the core logic
- a subset of the methods available update the methods and/or associated rights themselves; these methods are generally reserved for administrative roles

Email Objects

- From: `<email address>`
- Reply-To: `<email address>`
- To: list of `<email address>`
- cc: list of `<email address>`
- bcc: list of `<email address>`
- Subject: `<string>`
- date: `<timestamp>`
- body: flat file, eg. Html or Text
- attachment(s): files
- ueid: `<hash>` of other fields

Email Event Objects

- Application_id: `<email address>`
- input_ueid: `<hash>`
- output_ueid: `<hash>`
- transaction_id: `hash` of the concatenation of the input ueid and the output ueid

A Federated Network of Applications

CLAIM: agnostic to application composed with

CLAIM: recipts (aka email event objects) which are checksums for the computation of email objects allow pairwise verification of email payload by upstream and downstream email applications

CLAIM: a network of email applications following a protocol of pairwise payload verification forms a federated computation environment

Exclude the cryptographic claims but still keep them in mind for the future
CLAIM: a chain of receipts (sequences of ueids and tx_ids) represents a path through the computation graph which is verfiable (eg if you present a claim of what that message/transaction sequence was you can prove that it resolves to the same hash)

Composing Email Applications

example claims

- team inboxes + individual email addresses
- ...
- existing and planned mailscript features

Composing Email Apps with web 2.0 Platforms

example claims

- github integrations: automatically trigger build on failure
- various automation use cases already captured by mailscript team
- existing and planned mailscript features

  ...

Composing Email Applications with Cryptographic Receipts

example patterns

- introducing checksums and content-addressing
- push protocol: sending tx_id alongside any ueids
- pull protocol: where you store your tx_id for your whole history and return it on request (requests themselves are methods so you can give limited access based on access controls)
- beacon apps: specialized apps used to reduce the overhead of managing tx_ids for most apps while still making them available when needed. eg, an email app that collects and stores the recipts broadcast by other email apps. serves as a verifier as needed.

Composing Email Apps with web 3.0 Platforms

example patterns

- individual email addresses functioning as crypto wallets
- team email addresses as wrappers for smart contracts
- ...
- such middleware may be anchored by bi-directional identity mappings between email addresses and on-chain addresses; this is one example pattern
- the approach for business process automation is sufficiently general to allow mailscript identities to map to on-chain identities in multiple cryptographically secured run-times

Similarities Between Email and Blockchain-based Smart Contracts

The group identity is equivalent to a (DAO) smart contract address, and an individuals email address is equivalent to a wallet address, making the email equivalent to a web3 transaction (emails content is the event payload).

A smart contract is a shared database with state dependent rules for mutating it

it is comprised of

- a predefined schema,
- a unique identity,
- mappings to other unique identities,
- rights afforded to the those other unique identities to mutate the state of the datastructure according to predefined methods
- sets of predefined methods with rules about their valid inputs

Consider a process requiring some human input u, where there is some prior knowledge x with some output decision y.

Replace this example with a crypto workflow:

For example, the prior knowledge x is that a patient is to be proscribed Tylenol if they have a headache and the desired decision y is a quantity of Tylenol to be proscribed. The human input u represents the severity of the person's headache with $u = 0$ implying that $y = 0$ quantity of Tylenol is provided. On the other extreme $u = 100$ and $y = T_{\max}$, the maximum dose of Tylenol for that particular patient. A medical 'policy' could be automated to non-deterministic (discretionary) input $u \in [0, 100]$ and return $y \in [0, T_{\max}]$.

This is a very simple example but the family of all possible composed automations, including hybrids of human oversight and automation can be constructed as from the form $x^+ = f(x, u)$ where $y \subset x^+ = x \sqcup y$ is an output, u is an input, x is the prior state of a database. Note that x^+ is the posterior state of the database which includes the new peice of information y which was emitted as part of the automated event (aka transaction).

This same pattern can be used for examples form the "crypto" space where smart contracts automate processes for small teams of open source developers.

Examples to write:

- Extended example from an existing workflow in a smart contract
- Re-describing that example using only "email"

Needed Diagrams:

- breakdown of email App architecture
- composition of email applications with each other
- composition of email apps with web2.0 platforms
- composition of email apps with web3.0 platforms
- "journey" of data through a chain of automations
- chains of receipts for a chain of automations

Difference from Blockchain Networks

Property of Interest	Blockchain-based smart contract platform	Email Based Federated Network of Applications
Individual Identity Management	wallet addresses provide pseudonymous identities with state dependent permissions	emails addresses provide pseudonymous identities with state dependent permissions
Group Identity Management	Decentralized Autonomous Orgnaization smart contracts manage permissions within a group	Email Applications manage permissions within a group
Languages supported	domain specific langauge eg Solidity	Any web development stack, eg JavaScript, WASM, Python
Data Replication	all history of all event data on all nodes	local history and optional storage of event proofs for data and/or transactions
Computation Replication	all computations peformed by all nodes	local computation with optional pairwise validation of input-output validity
Hosting Infrastructure	Specialized Virtual Machines running in a distributed yet synchronized cluster	Local or cloud based software capable of connecting with an email server
Fees for use of platform	global compute is a scarce resource aquired via fees, eg Gas or native tokens	local compute is an abundant resource requiring nominal opex costs rather than platform fees
Other Economic Factors	Algorithmic Monetary policies manage supply of native tokens, adding operational complexity	no platform level economic requirements or capabilities, reducing operational complexity
Communication Protocol	Domain specific Machine-to-Machine communication protocol with limit support for applications and limited support for humans	Email is the de-facto communication protocol for Human-to-Human and Application-to-Human interactions in web2.0
Degree of Synchrony	Globally synchronized ordered list of transactions	Asynchronous, locally ordered list of events.

Conclusion

This document outlines the merits and use cases for a federated network of Email Applications, which include a range of use cases often attributed to blockchains and smart contracts. These use cases focus on applications which benefit from an ease of use increase and are not particularly sensitive to reductions in algorithmic regulations. For most use cases, the extreme degree of algorithmic regulation in blockchain based smart contract platforms places unnecessary burdens on application development. However, there remain a range of use cases for which this algorithmic regulatory framework is worth the overhead, for example: managing financial transactions, digital asset management and governance at large scales. While the earliest applications of smart contracts have focused on high value and large scale processes, the high degree of regulation has lead to a choice between low user experience or reintroducing intermediaries. Therefore, it stands to reason that email based applications and blockchain based smart contracts are complementary technologies wherein the smart contracts manage the processes which demand the most regulation whereas the email based applications can serve most users everyday needs.