Identity Agents

Paul Trevithick, The Mee Foundation

March 20, 2023

Abstract

This document describes symptoms caused by today's power imbalance between relatively powerful digital service providers and mere users. We then present a set of design considerations we think need to be addressed in any potential solution. Finally we propose an architecture for an *identity agent* that represents the user and protects their interests online along and working in concert with an associated legal contract.

1 Power imbalance

While the internet has brought new services and experience to billions of users, it has also resulted in a power asymmetry between the digital service providers and their users regarding these users' digital identities and personal data. These providers have accumulated power relative to their users.

The internet's designers endeavored to create decentralized architectures that pushed computation and storage to the edge and minimize centralized control. However, in the last couple decades economic factors and so-called natural monopolies, preferential attachment network effects, economies of scale, and the relative ease of creating centralized solutions have all contributed to concentrations of power on the provider side.

When Berners-Lee created the web, it was a decentralized platform. Anyone could publish a website and link to any other side. But as the web has grown from an obscure research-sharing community into a global medium for commerce, communication, journalism and entertainment, the power dynamics have shifted. Today, huge companies like Amazon, Meta, Google, and Netflix dominate the web. These corporate giants enjoy an enormous amount of control not only over what people see and do online but over users' private data.[1]

In response, many initiatives and projects are propose alternative approaches and tech-

nologies. Here are just a few examples: rececentralize.org[1], DWeb principles[2], The Web3 Foundation[3], the Decentralized Identity Foundation(DIF)[4], "local-first" software principles[5], ProjectVRM[6], Blue Sky[7], and the Decentralized Information Group[8].

To reinforce our focus on power relationships, we use the term *computational power* to refer to software tools that work (i.e. provide agency) "on the user's side" for, and *exclusively* on behalf of, the user. We are not talking about whether the average user's phones and laptops have sufficient computational power; they do. We're saying that this raw computational power and storage is not leveraged by tools that push back on app provider's power, and engage digitally and automatically to empower the user to better manage their relationships with providers on terms more beneficial to them. Since our discussion applies equally provider's mobile apps, webapps, and websites, we simply use the term *app* to refer to all of them.

Power asymmetry lies at the root of a diverse set of related symptoms, most of which erode privacy, since privacy and power are highly interrelated concepts[9]. These privacy eroding symptoms include a lack of personal autonomy, a lack of personal agency, and third-party surveillance.

1.1 Lack of Autonomy

au•ton•o•my: *freedom from external control or influence; independence.*[10]

Independence. In the physical world each of us is a separate, independent entity. Each of us has a self that embodies our individuality. We "bring" that independent selfness to interactions with others, with vendors, etc. while understanding that this independence is not absolute–we are still to some extent dependent on common, shared systems, laws, environments, and so on. By contrast, online it has been said that "we have no *digital embodiment.*"[11] Our identities are provided to us by digital service providers (e.g. in the form of a Facebook identity, or an Amazon account). Without them we don't exist. Anyone who has been banned from a platform, or uses a platform that is shut down is sharply reminded that their digital identity exists at the pleasure of that platform. Our provisional existence is the original power asymmetry. Efforts create personal datastores, or even more

[1] redecentralize.org
[2] getdweb.net/principles/
[3] web3.foundation/
[4] identity.foundation
[5] inkandswitch.com/local-first/
[6] blogs.harvard.edu/vrm
[7] blueskyweb.xyz/
[8] dig.csail.mit.edu
[9] Consider the title of Véliz's recent book, "Privacy is Power"[7]
[10] languages.oup.com/google-dictionary-en/
[11] Phil Windley, personal communication, Sepember 2022

to the point, those that strive to provide each of us a *self-sovereign identity*[6] are squarely aimed at addressing this issue–the word "sovereignty" certainly evokes power.

Ownership. Our personal data is collected and held by businesses (first-parties) as we interact with their apps, not by us. This pattern *app-held data* is so common that it's hard to imagine an alternative. Our data is not free from external control by apps, because it is generally stored and managed by them.

Our data is also collected and held by third-parties (e.g. data brokers) with whom we have no interactions at all. In short, it's been said that "everybody has our data ... except us."[12].

As we'll discuss more later on, there are alternative approaches. One is "user-held" data[2], where your data is held by you in a personal datastore)[13]. Another is following "local-first" software principles.[4]

Lock-in. As we've just mentioned our online existence is provisional. Further, this existence is bound to the provider from which it originated. Providers hold our data, and although in many jurisdictions providers are required to allow us to have access (e.g. to request that we can download a copy), we lack the technical means to accept the data stream and hold (e.g. in a personal datastore) that has the ability to subsequently transform it into other formats and schemas so as to make it reusable in other contexts. This lack of agency results in our data being held hostage, i.e. without autonomy.

Peer-to-peer With a few exceptions, e.g. technologies like Berty[14], internet users, when they communicate person-to-person don't have the ability to do so *peer-to-peer* from their edge devices to the other person's device. Instead, they are dependent on servers hosted by intermediaries. Whereas these days the messages themselves are end-to-end encrypted, the metadata (e.g. who a person communicates with, from where, at what time, how often and from which device, etc.) is in many cases visible to the intermediary's server.

1.2 Lack of agency

a•gen•cy: *the capacity, condition, or state of acting or of exerting power* [15]

Wielding credentials. In the offline world you can autonomously present your drivers license to a wine seller in order to prove that you are of drinking age since the wine seller trusts the license issuer. The interaction is privacy-respecting because the presentation interaction is

[12] reb00ted.org/personaldata/20210620-who-has-my-personal-data/

[13] Examples of open-source personal datastores include `https://solidproject.org`, Decentralized Web Nodes(DWN). For more about personal datastores see `https://wikipedia.org/wiki/Personal_data_service`

[14] berty.tech

[15] www.merriam-webster.com/dictionary/agency

not disclosed to the issuer. This could be described as "wielding" a trust credential. At present, there is no equivalent way to do this online. There's no standard way to be issued a credential, hold that credential in digital wallet, and then present that credential. With a few, domain-specific exceptions (e.g. cryptocurrency), there is no common, generalized method for you to prove something about yourself as stated by one party about you, to another party online.

Data presentation. One reason for form filling and other kinds of data entry on providers apps/sites is that the user, even if they were equipped with a personal datastore, lacks the ability to present personal information digitally to the provider. Instead the information must be re-entered manually at each provider. The credential presentation interaction mentioned above provides specialized example.

Delegation. In the offline world one entity can grant access to some resource to another entity. For example, I could give my car keys to a friend so they could borrow my car. There is no standard, or secure way to do this online. This is especially problematic in healthcare scenarios where a caregiver needs to gain access to electronic health-related data about another person.

Provider-defined Privacy Policies. Users are given the option to review the privacy policies put forth by the provider, policies which are designed to protect the provider's interests while staying within the limits defined by the relevant privacy regulations. The burden of making sense of these policies is shifted to the user (i.e. the potential victim) because the user doesn't have the time to read 100+ policies for the providers they typically use, nor do they have the computational power on their side to aid them in this assessment.

Privacy policy expression. With a few exceptions, (e.g. the Global Privacy Control[16]), users lack the technical means (i.e. computational power) to express their privacy terms to providers.

User rights. In a growing number of jurisdictions, starting with Europe's GDPR and expanding to other regions, the user's data rights, (e.g. the right to access, correct and delete their data), are explicitly stated. In principle these laws respect these rights, however in practice the time and effort required to exercise these rights is so exorbitant, that in practice they don't exist. The user has to send written requests to get their data, request that it by updated or deleted, etc. User-side agents are required for user's to regain in practice the rights they have in principle.

Inferences. "Eli Pariser need to give users control of their bubble" check this quote[5, p66](interests profile)

Feudalism. [You (as a peasant) work the fields, the landlord owns them and amass fortunes; mentions of Jared's and others AI injustice about who owns the work. Jared Lanier,

[16] globalprivacycontrol.org

etc.]

1.3 Third-party surveillance

Whereas the user is at least aware when they sign up on a first-party app that their interactions are known to the provider of that app, there are hundreds of third-parties of which the user is unaware that track and assemble databases about them. Databases of user data in the hands of hundreds of unknown third-parties creates privacy risks and vulnerabilities. Users have little transparency into what's being gathered, where it's being shared and how it's being used. It is worth noting that that much of this third-party tracking is enabled in collaboration with first-parties (e.g. first-parties placing third-party tracking cookies on the user's browser).

Surveillance-based targeted advertising. Targeted advertising in general[17] involves four main processing steps: (1) the collection of observations about the user by a first- or third-party, (2) synthesis of an "ad profile" from these observations, (3) matching this ad profile against available "target audiences"(i.e. characteristics of whom the advertiser wishes to reach, advertising budget, etc.) from advertisers through a bidding process, and (4) displaying the winning ad. Surveillance-based targeted advertising specifically is when step (1) above is achieved by third-parties who track the user as they move from apps to app and site to site across the internet using third-party cookies and similar tracking mechanisms.

Third-party ad tech vendors perform the tracking and synthesis of a user's "ad profile." Users have no say in their own ad profiles–never seeing them and not the ability to correct them.

Data brokers. Data brokers who buy and sell personal data to other brokers, to advertisers, adtech firms and first-party publishers provide liquidity (along with a host of privacy threats) in the personal data shadow marketplace because user's lack the computational power to provide data about themselves.

1.4 Inconvenience

The prevailing architecture of the internet involves each provider managing their own information "silo" of information about the user (i.e. their account). This approach and the lack of computational power on the user's side creates inconvenience for them that is described below.

Repetition. When using apps, users are often asked to provide information about themselves that another app has already asked them such as "what is your email address?" This is a symptom of the internet's silo-ed architecture wherein each app maintains its own

[17] Also known as behavioral advertising or more recently, interest-based advertising

database of personal information. The user has the hassle of repeated data entry, the app has increased friction (a worse user experience).

Password management. The average user uses 100 websites and 25 apps daily. Managing and periodically updating strong, unique passwords at each is impractical without an automated password manager (computational power), yet it has been estimated that less than 5 percent of internet users use a password manager.

Account Management. The user has the inconvenient burden of maintaining the timeliness and consistency of their account information at over one hundred apps. For example, updating contact or credit card information at each is tedious, time-consuming and encourages the user to spend more time at sites that already have their information. The relative convenience of shopping on Amazon vs. another e-commerce site. It is partially caused by the user's lack of computational power to manage these relationships–the processes are not automated and tedious.

2 Design Considerations

Our vision is to develop an identity agent to address the problems outlined in the previous section. This agent can represent the user and promote their interests online. In this section, we discuss a number of design considerations for this agent illustrating the each with use cases.

2.1 User-centric vs. provider-centric

Many of the challenges described thus far have their origin in an architecture that is *provider-centric* rather than *user-centric*, or *human-centered*. In the provider-centric model each provider sees a single narrow slice of the user through the lens of their direct interactions with them. Each works in isolation to optimize the user's experience on their app or at their site. To the user the situation is reversed. They sit at the center of many dozens of connections radiating out from them to apps/sites. The user has the burden for entering, and updating information at each provider each of which maintains a separate copy.

The best technology so far to address this user burden is a browser form-fillers which greatly reduces the number of keystrokes required to fill in web forms. Browsers are *user-agents* that perform this operation on the user's behalf, and sit on the user's side of the power equation. Another closely related user-agent technology is that of password managers. In both cases these agents maintain small databases on the user's side.

2.2 Edge-centered vs. cloud-centered

Given that we need a per-user, user-centric decentralized architecture, where should this datastore live? We look at two options edge-centered and cloud-centered. Edge-centered

means that the primary location for a user's personal datastore is on their own phones, and laptops and perhaps home servers. Cloud-centered means that the user's personal datastore is primarily held in the cloud (e.g. on a SOLID[18] pod). We say *primary* because there are usually use-cases that involved replicating/syncing some of the data to the "other" location.

Security. Although some may disagree, it is our contention that having a personal datastore on a personal device is more secure than in the cloud. Even if each platform alternative where equivalently secure, a cloud-centered architecture aggregates millions of personal datastores at one service provider and thereby creates millions of times the economic incentive for hackers to invest in attacking it.

Equity. The hosting costs of a cloud-centered solution must be paid for by some entity whereas user's typically own their edge devices and they are thus on a marginal basis "free". By equity we mean here that we need a solution that can be afforded by all socio-economic classes, and a solution that requires monthly hosting fees can thereby be ruled out.

Backup. One serious drawback of what's called a *non-custodial* edge-centered architecture (when compared to cloud-centered) is the need for the user's data to be backed up. This is not a problem if the user backs up their devices (e.g. to a cloud backup service), but many can't be relied on to be disciplined about this. [stats on phone vs. laptop backup]. [we assume the user has N¿1 device and that the agent is installed on N¿1 and that the agents replicate/sync]

2.3 Replication

If we assume an edge-centered design, we must solve the roaming problem. That is, we must support use cases where the user has more than one device and needs to be able to pick any one of them and have their person datastore be consistent across these devices (at least eventually). This requires that the agents of a given user implement replication and syncing. [discuss the (unfortunate and costly) need for relays to achieve P2P sync in some use cases].

2.4 Loyalty

Much of the power asymmetry described in the first section is due to economic incentives for providers to do just enough in the user's interest to keep them as a user or customer, but not more. Personal data, after all, is considered by business to be an asset class; the more of it that is collected and monetized the better. To have an agent that works not partially but exclusively on behalf of the user, the agent provider must not have an economic incentive to provide anything less but complete loyalty to the user's interests.

[18] solidproject.org

Although there are other potential solutions (e.g. data cooperatives and data unions) we think the simplest approach is that the agent developer be a nonprofit organization that has no economic incentive to be anything but loyal to the user and to have no economic interest in their data. In fact there's no reason that that the agent developer to access or store the user's data.

2.5 Separation of app from db

[Talk about this from the point of view of the SOLID project or go-peer?? or ??. Also talk about how this is inevitable. Desktop OSes had it, then in 2007?? iPhone got it, now in 2023! webapps will get it. Talk about automatic data portability]

2.6 Delegation

[Talk about delegated medical records use case]. [Talk about the Gropper Principle]

2.7 Multi-contextual

Zuckerberg has said "Having two identities for yourself is an example of a lack of integrity"[3]. He could not be more wrong. In everyday life people indeed have one *selfness*, but they have many, context-dependent *whonesses*.

2.7.1 Selfness and Whoness

In his last public speech[19] Kim Cameron[20] introduced two useful definitions based on archaic English:

- **Selfness**: The sameness of a person or thing at all times or in all circumstances. The condition of being a single individual. The fact that a person or thing is itself and not something else. Individuality, personality.
- **Whoness**: Who or what a person or thing is. A distinct impression of a single person or thing presented to or perceived by others. A set of characteristics or a description that distinguishes a person or thing from others.

The following diagram illustrates these concepts and introduces the notion of context:

[19] www.youtube.com/watch?v=9DExNTY3QAk
[20] en.wikipedia.org/wiki/Kim_Cameron_(computer_scientist)

2.8 Open source

In order to trust that the agent does what we claim, we need transparency which open source can provide. Further this is an ambitious project and we need to nurture the creation of a community of developers help build it.

2.9 Data Governance

Once data is shared from the agent to a first-party there are no technical means to constraint what the recipient does with it. No technical means can prevent them from selling it others, for example. Instead, legal means must be employed. Rather than wait for privacy regulations to get strong enough, we propose that first-parties sign a Human Information License to license the user's information under terms that are fair and balanced and respect the user's privacy rights. This contract can be signed by an entity that represents the user and makes it effortless for them.

2.10 User rights

[to be written: even the best privacy legislation while good in principle, does not in practice protect users because these laws don't mandate associated automated technology-based means for their implementation.]

2.11 Enforcement

[to-be-written: Talk here about the need for an entity to audit and enforce compliance with the legal contract]

2.12 Privacy by design

[to-be-written]

3 Identity Agents

3.1 End-user perspective

An *identity agent* is an app that gives the user control (power) over their own personal information as they interact with websites, mobile apps, and other user's agents. It does this through a combination of technical and legal mechanisms.

3.1.1 Privacy and Autonomy

The agent runs on the user's edge devices (mobile phone, laptop, etc.) where, entirely under the user's control, it holds a local, private database of the user's personal information. When an app/site wants to know something about the user, the agent shares as much or as little as the user chooses. If an app/site signs a Human Information License and thereby becomes *certified*, the legal entity behind that app/site is thereby obligated to (i) require explicit consent for collection, processing, storage and sharing of the users data and to (ii) implement APIs to exercise the user's rights to access, correct and delete the user's personal information. We envision that agents could provide ad profiles to certified publisher websites that are supported by interest-based advertising while eliminating the need for surveillance by third-parties.

3.1.2 Convenience

Although the agent is an interactive application, it operates in the background most of the time. Always working solely in the user's interest, it collects information from sites that already hold their data, and shares it with other sites that need it. Our vision is that that the user *never has to repeat themself* (nor remember passwords!) as they move from app to app and site to site across the internet.

Here are a few examples. If a site wanted to know the user's email address it might ask for it in a web form. In this case the agent would use it's form-filler "protocol" to fill in the value. If the site supports password-less sign-in (e.g. using OpenID Connect) the agent acts as the identity provider. If a site needed a digital driver's license credential, the agent acts as a digital wallet and presents this credential that it had presumably downloaded

earlier from an issuing site. In these different examples, different protocols for information sharing would be used, and the agent must be technology agnostic and support all of them. Only in this way an an agent be human-centric and put the one user at the center of all of their online relationships.

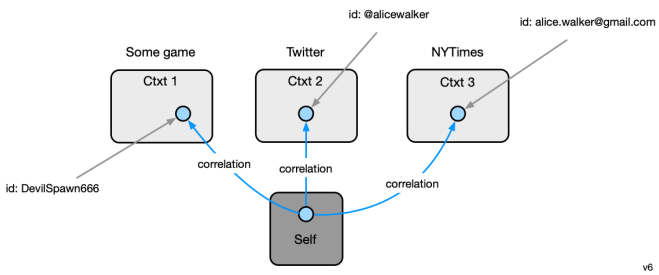
3.2 Self and contexts

The agent represents both the user's single *selfness* and their multiple context-dependent *whonesses*.

The selfness of the user is held in a data container called the *self*. The contents of the self are holistic and therefore quite sensitive. For this reason they would normally not be shared in a direct or comprehensive form with others. The user's self is the point of integration across contexts each of which may be from differing identity systems, use different protocols for communication and different schemas for knowledge representation.

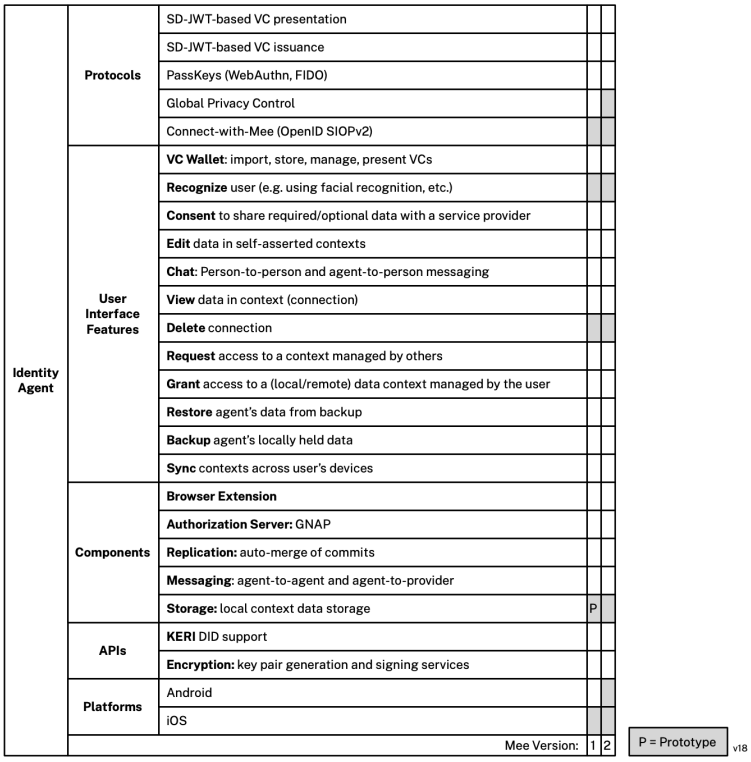
Each context is represented by a *context* data container. A directed *correlation* link points from an entity in the self to the entities representing the user in each context. To ensure privacy only the user knows that each of these separate contexts contain representations of them. Each context represents an interaction via some communications protocol with an external app, website or agent.

We can illustrate these concepts with a simple example. A user might play a game on a gaming site using the id DevilSpawn666, while communicating on Twitter as @alicewalker and subscribing to the New York Times as alice.walker@gmail.com. Here's a simplified view of how this is represented:



3.3 Functionality

Here is a summary of the functionality of an example agent.



3.3.1 Protocols

- SD JWT-based VC presentation
- SD JWT-based VC issuance
- PassKeys (WebAuthn)
- Global Privacy Control
- Connect-with-Mee: (OpenID SIOP) that uses a universal link to the agent

3.3.2 UI Features

- **VC Wallet:** import, store, manage, and present Verifiable Credentials (VCs). Note: the [OWF conceptual architecture](https://github.com/openwallet-foundation/architecture-task-force/blob/main/docs/architecture/conceptual-architecture.md) adds Burn, Receive, Send, Transfer, Refund, Purchase, Withdrawal, Deposit
- **Recognize** user (e.g. using facial recognition, etc.)
- **Consent** to share required/optional data with a service provider

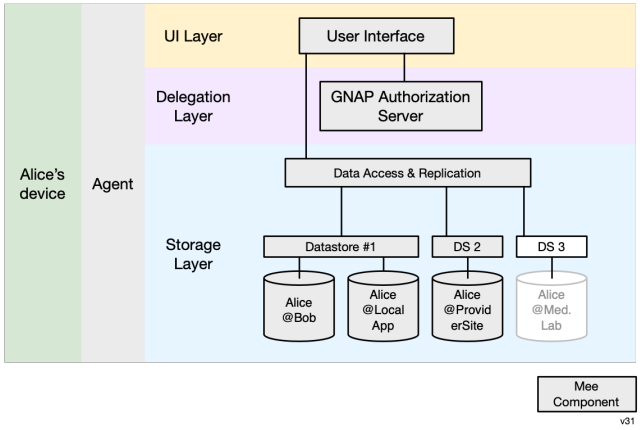
- **Edit** data in self-asserted contexts
- **Chat:** Person-to-person and agent-to-person messaging
- **View** data in contexts
- **Delete connection** delete all data associated with this set of contexts
- **Request** access to a context managed by others
- **Grant** access to a (local or remote) data context managed by the user
- **Restore:** recover all data using SRP and backups
- **Backup** local contexts
- **Sync** contexts across user's devices

3.3.3 Components

- **Authorization Server**: GNAP AS
- **Replication**: synchronization of context state across the user's set of agents
- **Messaging**: communications among members of the user's set of agents
- **Storage**: local data storage for contexts

3.4 Architecture

In this section we propose an architecture for identity agents. We consider a user, Alice, and her agent's three layered architecture shown below.



3.4.1 UI layer

Alice's identity agent is deployed as an app on Alice's device (e.g. a smart phone). The top, UI layer provides Alice with data management features to connect with apps/sites and manage her data. This UI allows her to inspect and in some cases edit each of the partial representations of her in each connection's context(s). Part of this UI is an "agent facade"–a presentation layer for how Alice appears to other agent users.

3.4.2 Delegation layer

The GNAP[21] Authorization Server (AS) responds to requests for access to Alice's context datastores wherever they may be located physically. These requests could come from any app/site whether local or remote. In other words they could come from local apps (e.g. Provider's App), remote app/sites (e.g. Provider's Website) or from other users' agents (e.g. Bob's Agent). The AS is integrated with the agent's UI to allow Alice to grant or deny these requests for access. If Alice, either through explicit UI interaction or via a policy she has established, grants the request, the AS returns an access token to the requester. The requester presents this access token to a context datastore when it needs access.

3.4.3 Storage layer

The Data Access and Replication component has two responsibilities. First, it provides an abstraction layer performing schema mapping such that the UI layer with which it is integrated has read/write access to context data in an abstract, universal schema. This is the "data access" responsibility. The second responsibility is to manage the replication of data within Alice's context datastores across all of Alice's devices (phone, tablet, laptop, etc.).

Below the Data Access and Replication component lie a set of datastores each using its own datastorage technology to hold one or more contexts (data containers). Each context holds a contextualized representation of Alice as defined (as to schema) and created and managed by apps/sites. The diagram above shows three local contexts on Alice's device and one, the Med Lab app's context, which is not replicated on Alice's local device (perhaps because its data set is too large for Alice's device).

In this example, Bob's agent is accessing a context called "Alice@Bob" as managed by datastore 1. Provider's Local App is accessing a context called "Alice@Local App" that is also managed by datastore 1. An external Provider's Website is accessing a context called "Alice@ProviderSite" that is managed by datastore 2.

[21] oauth.net/gnap/

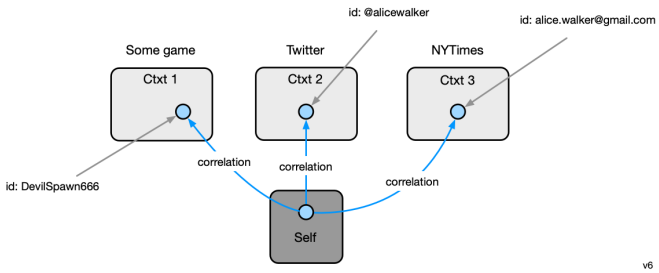
3.5 Data model

This section describes the data model of the agent. The user's data that follows this model is replicated across instances of their agent running on different devices. At the highest level, the data model can be thought of as a three level hierarchy of data containers (*Container* subclass instances) each of which holds *Person* instances representing the user. The top layer consists of a single *Self* container. The middle layer are *Group* containers. The bottom layer consists of *Context* containers.

These Person instances are connected into a directed graph that spans these three levels of containers. The singleton Self container holds a single Person node that represents the selfness of user as a single individual. The Self has a set of Context containers each of which represents how the user is presented to or perceived by another party (e.g. another person's agent or a digital service provider's app/site)–that is their whoness. Note that any number of combinations of communications protocols, local apps and web services may be involved in the connection between the agent and another party. The Person node in the Self container has no scalar attributes but usually contains a set of correlation links pointing to a corresponding Person node (representing the user) in each of N contexts.

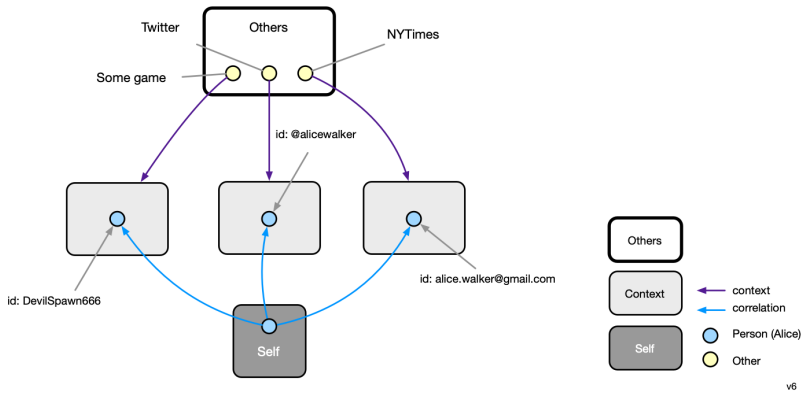
Between the Self and the leaf Context containers may exist a set of intermediate level Group containers. These also contain a Person node representing the user. This Person node is linked to "sub" Person nodes in the child containers of the Group container. It may also have attributes of its own. The Person node in a Group container can be used to represent a role the user might play in a set of child contexts.

In the simplified example below we show a user, Alice, whose selfness is represented by a blue Person node in the Self context. Alice has a relationship with three other parties: a game, Twitter, and the New York Times. Each of these relationships is represented by a context. The whoness, or facet of Alice that she exposes in each context is represented by a Person node in each of these three contexts.



The information in a context (most importantly person nodes) is read and written to by the agent based on the data flowing through the agent's connection with the other party (or more precisely, with the apps and websites of the other party). We have added these

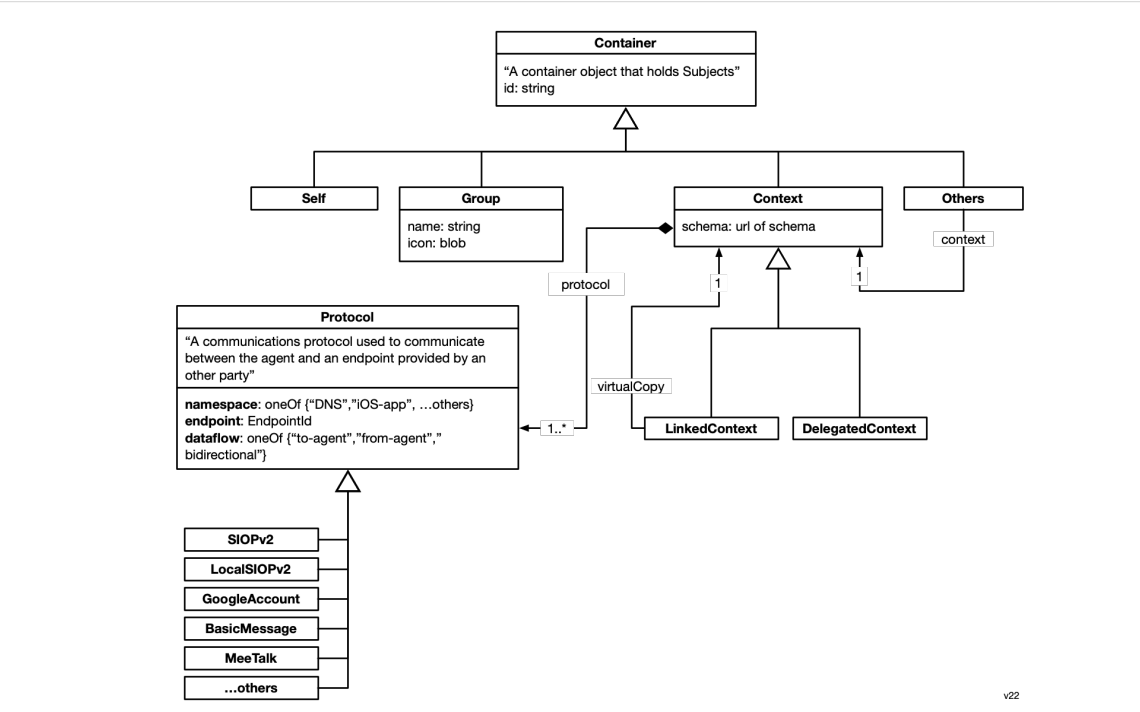
other parties explicitly to the diagram below by introducing a new kind of context called Others within which are objects representing other parties.



The personal data flowing through each of the three connections represented by the purple lines above may flow from the agent, to the agent or in both directions. It may have originated on either side. It may be self asserted claims (attributes) entered by the user directly into the agent. Or it may be claims entered by the user on a website of the other party, or sensed by a local app (or sensor), or generated by the other party based on direct on-site or on-app interactions with the user.

3.5.1 Container classes

We describe the data model in two parts. The first part describes the data containers. The second describes the data that is held by those containers. Let us start describing the data model of the containers themselves. Here are the various data container classes:



Classes

- **Others** - a container holding a set of Other nodes (see Persona Classes). Each Other node represents a party with the user has a connection. These Others may be other people or legal entities. If they are legal entities, they are often called relying parties, such as a digital service provider like Twitter, Inc. Each other has the following properties:
  - **Context** - a single Context that captures one aspect of the overall connection
- **Self** - the single Container holding a single Person node that represents the selfness of the user
- **Group** - an intermediate level container that holds a single Person node that represents a common role or persona that the user plays. A group has these attributes:
  - **name** - the name of the group
  - **icon** - a icon for the group
- **Context** - a Container holding a Person node that represents the user in a specific aspect of their relationship with some other party. We say "specific aspect" because the relationship between the user a given other, may be represented by more than one context, each representing a different aspect.

More about Contexts

A context has the following attributes, that taken together uniquely identify the context:

- **schema** - url of the schema of the data in the context
- **protocols**[] - array of one or more Protocol instances

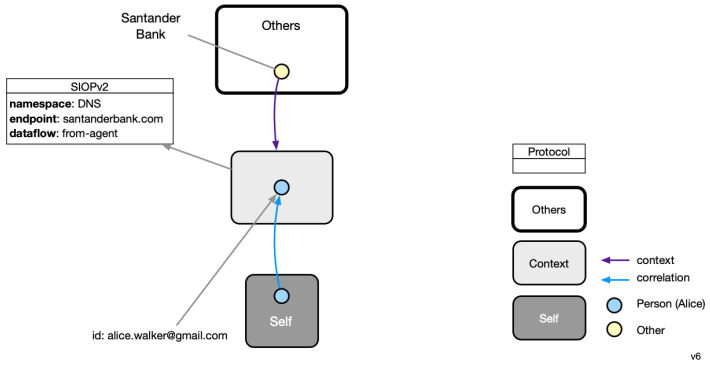
The kinds of data held by a context depends on the communications protocol (using the term loosely) between the agent and the other party. As will be described next, a Protocol class within the agent represents these data conventions using a schema that is an extension of the Persona schema.

There are two subclasses of Context: *LinkedContext* and *DelegatedContext* that are described in their own sections below.

Protocols

A Protocol class represents a communication protocol used between the agent and an endpoint provided by an other party. Each protocol subclass represents a different communications protocol such as SIOPv2, GoogleAccountSync, BasicMessage (DIDComm), etc. Protocol classes have a class method that returns the data schema used when it updates data in that context. These schemas are resolvable from a URL which is written to the *schema* attribute of the Context instance.

A Protocol is an attribute of a Context, and although a less common situation, may have more than one. Below is an example of a user Alice who has a (hypothetical) connection with Santander Bank. This connection has a single context that contains the information that Alice shares in with the bank via the OpenID Connect SIOPv2 protocol.



Each protocol instance has these attributes:

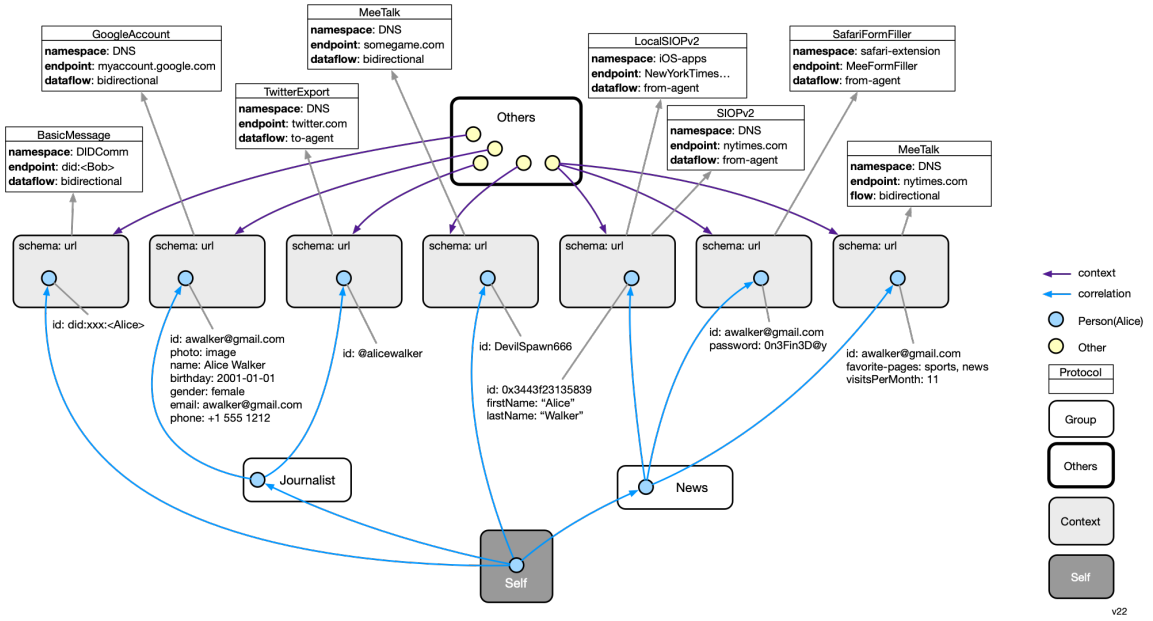
- **namespace** - a string that indicates the namespace used by the "endpoint" attribute

- **endpoint** - a string identifier that unique identifies the other party with which the user has a relationship within the above namespace attribute
- **dataflow** - one of to-agent, from-agent, bidirectional - indicates the direction of data flow between the agent and the endpoint

Multiple connections

In the example below, we expand our story about our user, Alice. She has defined two groups. The first represents her role as a Journalist, and it contains two contexts: the context representing her relationship with Google and with Twitter. The Google context contains her Google account profile which can be updated either using her agent or via the Google website (hence the "bidirectional" dataflow). Her Twitter context contains a snapshot of all of her Twitter account information, lists of who she follows, etc.

The second group, entitled "News" contains a Person linked to three context all belonging to the New York Times. The first of these three is the context that she uses, via SIOPv2 to login to the NYTimes website. The second is a context that contains data her form filler Safari extension uses. The last is a context that establishes a bidirectional connection with the NYTimes using a new (an purely hypothetical for now!) bidirectional data synchronization protocol called MeeTalk. She plays a game for which there is a context (without being within an intervening Group), and she has a direct relationship with her friend Bob using the DIDComm BasicMessage protocol.

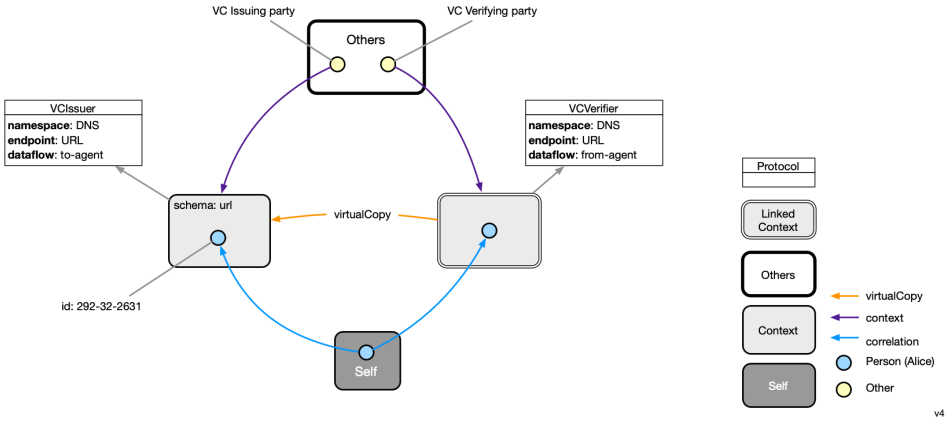


A relationship between the identity agent and another party is called a *Connection*. It is

represented by one or more other contexts each of which has a protocol (and sometimes more than one). Alice is shown with five connections–one for each of the five Other nodes in her Others container.

Linked contexts

Alice can convey claims made about her by one party and present them to another party. For this example we'll assume that the claims are encapsulated within a Verifiable Credentials[22] document.

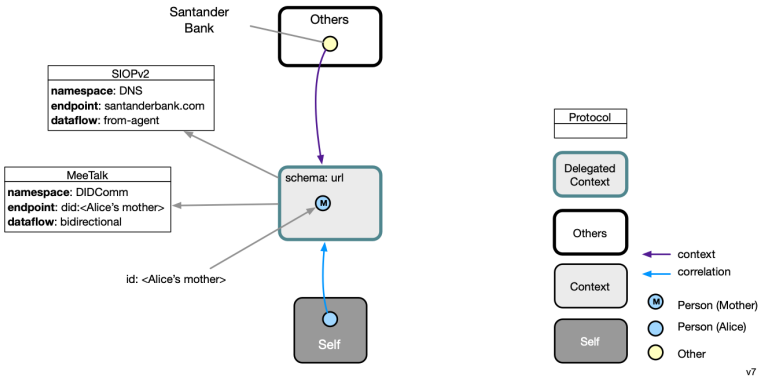


In this example Alice, presumably having authenticated herself in her connection to a site/app of the "VC Issuing party" is issued a VC containing claims about her which is stored in the leftmost context above. She then goes to another site/app of the "VC Verifying party" and finds that they trust the issuing party and would accept a VC issued by them. In Alice's second connection a VC presenting protocol is used to send the VC to the verifying party. This second connection involved a special class of context, called a *LinkedContext*.

Delegated contexts

Alice takes care of her elderly mother, and helps her mother manage her bank account at Santander Bank. Alice's mother has an agent and has delegated access to Alice a context that contains her mother's OpenID Connect SIOP claims.

[22] w3.org/TR/vc-data-model/

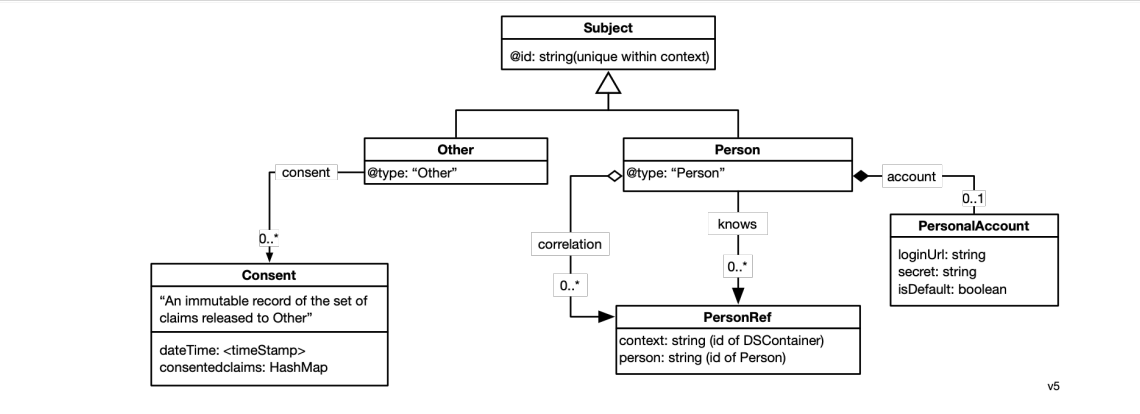


As shown above, Alice has a direct connection with the bank (that communicates via the SIOPv2 protocol) but the context for that connection is linked to a context managed by Alice's mother's agent. The MeeTalk data replication/sync protocol is used to ensure that Alice's DelegatedContext is always synchronized with the "original" context on her Mother's agent.

3.5.2 Persona classes

Group and Context containers all contain information about subjects (things) that are described according to the persona schema. [We need to decide on the Persona Schema Description Language (SDL) and describe the concepts below using it]. In knowledge representation parlance, the Persona schema would be known as an *upper ontology.*

In the Persona schema, people are be represented as instances of Person, A PersonalAccount class is also defined. These classes are shown below.



Classes

- **Subject** - kind of digital subject about which the agent stores information
- **Person** - a natural person, a subclass of Subject. Each person has the following properties:
  - **claims**[] - a set of zero or more properties. Here are a few examples:
    * givenName
    * familyName
    * phoneticGivenName
  - **account** - an optional PersonalAccount at some other party's site or app
  - **correlation** - zero or more PersonRefs that act as a link to a target Person object representing another whoness of the link's source's person's selfness.
  - **knows** - zero or more PersonRefs that link to a Person representing some other person (other than the user)
- **Other** - a Subject representing another person or a legal entity with which the user has a connection. Each Other object has:
  - **consents** - zero or more Consent objects. Each Consent has:
    * **dateTime** - time stamp of when the user consented to share this set of claims
    * **claims**[] - a set of zero or more claims (note: claim types (e.g. "email address") not their values)

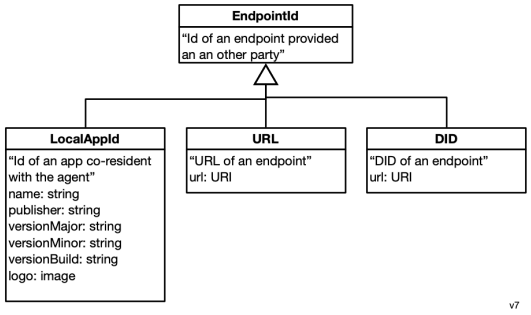
Extensions

Each protocol class will extend the Persona schema by defining Person subclasses, other new object classes and new kinds of relationships. For example the Google Google Account[23] API includes (optional) claims of "name", "gender" and "birthday". The protocol that supports the myaccount API would define these claim types in its schema, and insert a link to this schema in its corresponding context's *schema* attribute.

3.5.3 Datatypes

This section is largely incomplete, but will eventually describe lower level classes that we call datatypes that are used by the higher level classes mentioned above.

[23] myaccount.google.com



- **EndpointId** - an identifier of an endpoint (e.g. webservice or a local app) supported by an other party.
- **LocalAppId** - A specific kind of EndpointId. Uniquely identifies a service provider's mobile app.
- **Secret Recovery Phrase** - a 12-word textual phrase that the user creates. It is used to generate cryptographic keys that in turn are used to encrypt the user's personal data whether it is stored locally on their device or in a backup location. It can be used to generate keys to digitally sign transactions (e.g., for crypto currency transactions). It should never be shared with anyone or any service provider. If the user loses this phrase, they lose the ability to decrypt their data.

4 Agents and apps/sites

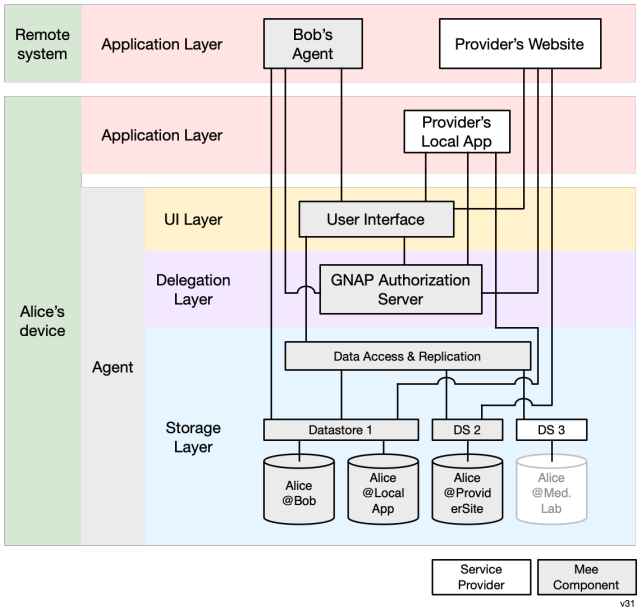
We now consider how an identity agent interacts with apps/sites. The diagram below shows Alice's agent from the previous section with the addition of a local as well as a remote application layer.

4.1 Interactions

The local application layer contains applications that run on the same device as Alice's agent. The digram shows one such application labelled "Provider's Local App". The remote application layer contains applications that run on systems and devices external to Alice's device. Two examples are shown. The first is Bob's Agent which appears to Alice as an app (just as Alice's agent appears as an app to Bob's agent). The second is a website labeled "Provider's Website".

The first step setting up a connection with a provider's app/site is for the app/site to authenticate the user's agent. We propose that the app/site implement the OpenID SIOPv2[24] for this purpose.

[24] openid.net/specs/openid-connect-self-issued-v2-1_0.html



4.2 Private data sharing

Data collected and held by the user's agent is clearly under the user's control. But data that they have shared with another party also needs to be under their control. We call this *private data sharing*. Since no technical means exist to control data transferred to another party, legal means must be used instead. The legal mechanism we propose is a "Human Information License" (HIL) contract between two parties. The first is the digital service provider legal entity behind a given app/site. The second is a legal entity, likely a nonprofit, that represents agent users. For more details see the "Human Information License" section.

The HIL imposes a number of obligations of the provider. Among them is the provider's requirement to respect the user's *data rights* to access, correction (editing), and deletion of the information collected and held by them. The user may have shared information manually (e.g. by filling in a form, or other kinds of interactions) or online from their agent. The HIL requires the provider to implement *data rights* APIs/protocols that an agent can use to remotely control this collected data.

These data rights APIs/protocols are an area of active research.

4.3 Human Information License

[to be written]

5 Contributors

Contributors to this document include Alexander Yuhimenko, Sergey Kucherenko, Kiril Khalitov.

References

[1] Klint Finley. Tim berners-lee, inventor of the web, plots a radical overhaul of his creation — wired. *Wired.com*, 4 2017.

[2] Paulius Jurcys, Christopher Donewald, Mark Fenwick, Markus Lampinen, Vytautas Nekrošius, and Andrius Smaliukas. Ownership of user-held data: Why property law is the right approach. *JOLT*, 2021.

[3] David Kirkpatrick. *The Facebook effect: The inside story of the company that is connecting the world*. Simon and Schuster, 2011.

[4] Martin Kleppmann, Adam Wiggins, Peter Van Hardenberg, and Mark McGranaghan. Local-first software: you own your data, in spite of the cloud. pages 154–178, 2019.

[5] Roger McNamee. *Zucked: Waking up to the Facebook catastrophe*. Penguin, 2020.

[6] Alex Preukschat and Drummond Reed. *Self-sovereign identity: decentralized digital identity and verifiable credentials*. Simon and Schuster, 2021.

[7] Carissa Véliz. *Privacy is Power: Why and how You Should Take Back Control of Your Data*. Random House, 2020.