Identity Agents

Paul Trevithick, The Mee Foundation

March 3, 2023. Revised April 15, 2023

Abstract

There is a power imbalance between digital service providers and their users, with regard to these users' personal information and online identity. The result is that users lack autonomy agency. They are subjected to third-party surveillance, and are inconvenienced by today's prevailing provider-controlled, siloed architecture for personal information. We explore design considerations for potential solutions and propose a design for an on-device *identity agent* and an associated legal contract, that together represent the user and protect their interests online.

1 Power Imbalance

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

The internet's designers, in the interest of resilience and scalability began with a decentralized architecture that moved computation and storage to the edge of the network and minimized centralized control. However, in recent decades economic factors, so-called natural monopolies, preferential attachment network effects, economies of scale, and the relative ease of creating centralized solutions have contributed to a concentration of power in the hands of a few relatively large providers.

Berners-Lee conceived the web as a decentralized platform wherein anyone could publish a website and link to any other site. He even envisioned that these pages would be editable. However, as the web has grown from a research-sharing community into a global medium for commerce, communication, journalism and entertainment, power has concentrated. Today, the web is dominated by enormous companies like Amazon, Meta, Google, and Netflix. These corporate giants enjoy enormous control not only over what people see and do online, but over their users' private data.[1]

In response, many initiatives have arisen proposing alternative approaches and technolo-

gies. Here are just a few representative 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 Berners-Lee's Decentralized Information Group[8].

We use the term *user agent*[9] to refer to software tools that work (i.e. provide agency) "on the user's side"[10] for, and *exclusively* on behalf of, the user. Since our discussion applies equally to a provider's mobile app, webapp, or website, from here forward we will simply use the term *app* to refer to any and all of them.

Power asymmetry lies at the root of a diverse set of related end-user symptoms, most of which erode privacy, since privacy and power are highly interrelated concepts[11]. 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.*[12]

Independence. In real life each of us is an 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, it has been said that "we have no *digital embodiment.*"[13] 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 has been 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 more specifically, those that strive to provide each of us a *self-sovereign identity*[9] are squarely aimed at

[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] developer.mozilla.org/en-US/docs/Glossary/User_agent
[10] Project VRM refers to this as "tools for individuals to manage relationships with organizations" to which we would add "...or with other individuals."
[11] Consider the title of Véliz's recent book, "Privacy is Power"[11]
[12] languages.oup.com/google-dictionary-en/
[13] Phil Windley, personal communication, Sepember 2022

addressing this issue–the word "sovereignty" certainly evokes power. Lanier refers to this as a person's right to a "First-class identity."[6, p251].

Ownership. Our personal data is collected and held by organizations as we interact with their apps, but not by us. This pattern *app-held data* by *first-parties* 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 direct interactions. In short, it's been said that "everybody has our data ... except us."[14].

As we'll discuss more later on, alternatives have been proposed. One is "user-held" data[3], where your data is held by you in a personal datastore[15]. Another is following "local-first" software principles.[5]

Lock-in. As we've mentioned our online existence is provisional. Our 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 store it under our control (e.g. in a personal datastore). If we did we'd have the potential to subsequently transform it into other formats and thereby make it reusable in other contexts. Thus our data is held hostage, i.e. without autonomy.

Peer-to-peer. With a few exceptions[16], internet users, when they communicate person-to-person don't have the ability to do so *peer-to-peer*–from their edge device to the other person's device. Instead, they are dependent on servers hosted by intermediaries. Whereas it is now standard practice that the content of messages is end-to-end encrypted, the meta-data about them (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 server.

1.2 Lack of agency

a•gen•cy: *the capacity, condition, or state of acting or of exerting power* [17]

Wielding credentials. In real life you can 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 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,

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

[15] 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`

[16] berty.tech

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

hold it in digital wallet, and then present it. With a few, domain-specific exceptions (e.g. cryptocurrency), there is no standard online method for you to prove something one party states about you, to another party.

Data presentation. One reason providers' apps rely on form filling and other kinds of data entry is that the user, even one 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 is another example of this.

Delegation. In the offline world one entity can grant access to some resource to another entity. For example, I can give my car keys to a friend so they can borrow my car. There is no standard, secure way to do this online. This is especially problematic in healthcare scenarios where a healthcare provider needs access to electronic health-related data about a patient, whereas the patient may not be able to provide it by themselves but instead needs to rely on someone else, e.g. a family member to grant the needed permission.

Provider-defined Privacy Policies. As is expected, providers define policies which are designed to protect the their interests while staying within the limits defined by the relevant privacy regulations. Users are indeed given the option to review these policies with the burden of making sense of them shifted to the user (i.e. the potential victim). Because the user typically uses dozens providers or more, they don't have the time to review each of these policies and they lack user agents to assist them.

Privacy policy expression. With a few exceptions, (e.g. the Global Privacy Control[18]), users lack the technical means (i.e. computational capability) to express their own privacy terms to providers.

User rights. In a growing number of jurisdictions, including Europe under GDPR[19] and California under CPRA[20], the user's data rights, (e.g. the right to access, correct and delete their data), are clearly described. In principle these laws respect these rights, however in practice the time and effort required to exercise these rights at every app/site is sufficiently onerous, that in practice they are not respected. The user must send written requests to get copies of their data, update it, or have it be deleted. The power of user agents is necessary to regain in practice, the rights they already have in principle.

Content filtering. Social networking platforms have replaced human content editors with algorithmic filters. Users might think that they see a balance of content whereas in reality they are trapped in what Pariser called "filter bubbles."[8] Pariser's recommendation is that if platforms are going to be gatekeepers, they need to program a sense of civic responsibility

[18] globalprivacycontrol.org
[19] gdpr-info.eu/
[20] thecpra.org/

into their algorithms, they need to be transparent about the rules that determine what gets through the filter, and "they need to give user control of their bubble."[7, p66]

Behavioral advertising. Behavioral, (aka targeted) advertising involves the automatic generation of interest profiles by third-party adtech firms. In most cases users have no control over these interest profiles, and no ability to correct them.

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 also 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 no visibility 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. Vanilla targeted advertising [21] involves four main processing steps: (i) the collection of observations about the user by a first- or third-party, (ii) synthesis of an "ad profile" from these observations, (iii) 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 (iv) displaying the winning ad. What many people find objectionable is *Surveillance-based* targeted advertising wherein step (i) 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, newer, cookie-less alternative identifiers, and other tracking mechanisms).

Data brokers. Data brokers are third-parties who buy and sell personal data to other brokers, to advertisers, adtech firms and first-party publishers. They provide personal data marketplaces behind the users's back. They could be considered an unfortunate, privacy-invading *iterim* solution to address the fact that users lack user agents with which to directly provide this data about themselves.

1.4 Inconvenience

In the prevailing architecture of the internet each ditial service provider manages their own information "silo" of information about the user (i.e. their account). This approach and the lack of sufficiently powerful user agents creates inconvenience.

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

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

symptom of the internet's silo-ed architecture wherein each app maintains its own database of personal information. The user has the hassle of repeated data entry and the app offers a less-than-optimal user experience.

Password management. The average user uses roughly 100 websites and 25 apps daily. Although managing and periodically updating strong, unique passwords at each is impractical without an automated password manager user agent, it has been estimated that less than five percent of internet users use one.

Account management. The user shoulders the burden of maintaining the timeliness and consistency of their account information at hundreds of 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. other e-commerce sites is a consequence partially caused by the user not having a user agent to manage these relationships.

Privacy self-management. Provider organization's privacy professionals publish policies to provide transparency as to what their apps do with user data. Their development teams implement privacy controls (e.g. cookie controls, and many other types) to allow the user to perform privacy self-management. As has been well documented, the resulting burden on the user across dozens of apps is unbearable.

2 Design Considerations

In this section, we discuss design considerations for a solution aimed at addressing the symptoms described in the previous section.

2.1 Human-centricity

Many of the challenges described thus far have their origin in an architecture that is *provider-centric* rather than *human-centric*. The internet is comprised of millions of providers, each offering their own app[s]. In this provider-centric model each provider's app sees a narrow slice of the individual through the lens of their direct interactions with that user.

For the individual the situation is reversed. They sit at the center of a hub with many dozens of connections to apps radiating outwards from them. Even for a single app there is considerable burden for the user to enter and update personal information, payment details, and preferences, and review privacy policies, and set cookie preferences, and so on. Multiplied by perhaps one hundred connections the resulting burden is completely unmanageable.

Tools to manage these chores must sit on the user's side and on their behalf across all

of them. Technologies of this kind, that empower the user across multiple apps, e.g. browsers and password managers, are called are *user-agents* since they act as agents of the user.

2.2 On-device storage and processing

If we assume a human-centric decentralized architecture, where should the user's personal datastore and associated processing live? Should it be on-device or in the cloud? By on-device we mean that the primary location for a user's datastore and processing is on their own phones, laptops, and perhaps home servers. Cloud-based means that the user's datastore and processing lives primarily in the cloud (e.g. on a SOLID[22] pod). We say *primarily* because there are usually use-cases that involved replicating/syncing some of the data to the "other" location.

Security. Although this is debatable, it is our contention that given a large number of users, having a personal datastore on-device is more secure than in the cloud. Even if each alternative were equivalently secure for a single user, a cloud-based architecture by its very nature aggregates large numbers of personal datastores at one cloud service provider location and thereby creates a much larger economic incentive for hackers.

Equity. An solution must be able to be afforded by all socio-economic classes and not just those better off. For this reason we believe solutions that incur monthly hosting fees are disqualified. Since users own their devices and can "host" new apps there, the situation is better, although there is a cost for the additional storage required for an on-device datastore.

2.3 Replication

If we assume an on-device personal datastore, we introduce the need to solve the related, so-called *roaming* problem. That is, we must support use cases wherein the user has more than one device that are intermittently connected to the internet, and needs to have their datastore state be consistent across these devices, at least eventually. This requires that the user's agents implement data replication and syncing between themselves. An open research question is how to achieve this replication without the need for relay servers and their attendant costs–costs which undermine the equity consideration previously mentioned.

Backup. One disadvantage of so-called *non-custodial*, on-device architecture (as compared to cloud-based architectures) is the vulnerability of users who are not diligent about backing up their devices (e.g. to an online service) to losing their agent-managed personal data. For users with more than one device this is less likely since data is replicated (as

[22] solidproject.org

mentioned above) across their devices, and a repaired or replaced device's data can be restored from one of the user's other devices. There remains of course the worst-case scenario wherein the user hasn't backed up any of their devices and all of them are lost or damaged simultaneously.

2.4 Loyalty

Much of the power asymmetry described in the first section is due to economic incentives for providers to motivate them 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 and thus the more of it that is collected and monetized the better. If an agent is to work *exclusively* on behalf of the user, the agent provider must not have an economic incentive to provide anything less than complete loyalty to the user's interests.

Although there are other potential solutions (e.g. data cooperatives and data unions) one of the simplest is that the agent provider be a nonprofit organization that has no economic interest in the user's data. This being the case, there is no need for the agent provider to have any access to the user's data.

2.5 Metacontextual

Zuckerberg once said that "[h]aving two identities for yourself is an example of a lack of integrity"[4]. However, even if one could force all users of a single system (e.g. Facebook) to have a single identity, this approach is clearly unworkable for a solution that represents the user across multiple, widely varying systems and contexts. People need the freedom to be themselves–selves that are complicated and messy. Our identities vary depending on whom we are interacting. We choose to express different parts of ourselves within different contexts. Not only are the attributes we share different the values of one attribute may be different in different contexts.

> "[A]t various times in the same day, virtually every adult can be a friend, a worker, a supervisor, a citizen, a mentor, a student, a musician, a customer, a lover, a child and a parent. Each of these roles demands different behavior and different aspects of our selves, aspects that need not be consistent. We behave, for example, in different ways with loved ones than with those we encounter in commercial or professional settings. Even among our loved ones, we behave very differently (and often show very different sides of ourselves) to our children, our parents, and our sexual partners. But this is not dishonest, nor is it inconsistent. At the very least, it's no more inconsistent than is the complicated nature of having a self. It is human."[10, p122]

Let's look at a person's age as an example. We see that across contexts they might share, their exact chronological age among their close friends, a fictional age to a music recom-

mendation service, no age at all in contexts wherein doing so might cause discrimination against them, or a merely a statement that they exceed the legal drinking age.

In his last public speech[23] Kim Cameron[24] 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.

Figure 1 illustrates these concepts and introduces the notion of context.

Figure 1: Multiple whoness-contexts around a single selfness

Using these terms we can say that in everyday life people have one *selfness*, but they have many, context-dependent *whonesses*. Any solution must be metacontextual–it must embrace and explicitly support the complicated, multi-contextual nature of our lives.

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

2.6 Delegation

In *A Human Rights Approach to Personal Information Technology*[2] Gropper asserts that there is an architectural principle that must be adhered to in order to respect human rights [e.g. to privacy]. He identifies three universal components:

- **Authentication** (signing-in and signing documents)
- **Request** for information (e.g. forms, searches, conversations)
- **Storage** (e.g. labs, prescriptions, social contracts, transactions [, other human information])

The then asserts what could be called the *Gropper Principle* as follows (our words, his ideas):

> "Any system that respects the human right to privacy must not bundle authentication, request, and storage."

In his presentation[25] at the 2022 Identiverse conference provides additional detail (see slides[26]). It explains that only a decentralized architecture can implement the Gropper Principle because each of the three components needs to be implemented separately. For this to work in an open world with multiple alternative component providers, there will need to be a convergence on open standards between these three components.

2.7 Trustworthiness

Any agent solution for online identity is by it's nature managing highly sensitive information. In order to be adopted voluntarily by people any solution must be trustworthy–its users must have confidence that their information isn't being used against their interests.

Solutions based on open source software can leverage transparency to help build this confidence that the solution is trustworthy. In open source software the source code is visible to anyone to review and audit to ensure that the solution is secure, free from vulnerabilities, and works in the user's interest.

In addition to open source, users will also consider the nature of the organization offering the solution as to trustworthiness. The organization's financial incentives should be aligned with their users interests. In this regard a nonprofit organization can be created that has no financial or business incentive to exploit the user's data against the user's interest. Ideally the organization would have no need to have any access to the user's data so that there's simply no reason to have to trust them, their security infrastructure and practices.

[25] identiverse.com/idv2022/session/841489/

[26] drive.google.com/file/d/1lwaMVkG4kLi7z6cXhqMx-DGkUww9azW3/view

2.8 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.

2.9 User rights

People should enjoy user rights to access, correct and delete their own personal information managed by providers. Unfortunately, even the best privacy legislation while providing these rights in principle, does not do so in practice. The reason they fail in practice is that the combined burden of the mechanisms to exercise these rights across hundreds of apps, is unmanageable for the user. To provide these rights in practice, providers must implement APIs and user's must have user agents that can consume them on the user's behalf.

2.10 Enforcement

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

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 maintains 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. Figure 2 shows a simplified view of how this is represented:

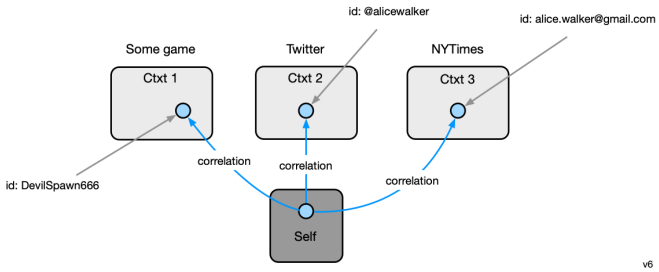

Figure 2: Alice with three contexts

3.3 Functionality

Figure 3 shows a summary of the functionality of an identity 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

Identity Agent	**Protocols**	SD-JWT-based VC presentation		
		SD-JWT-based VC issuance		
		PassKeys (WebAuthn, FIDO)		
		Global Privacy Control		
		Connect-with-Mee (OpenID SIOPv2)		
	User Interface Features	**VC Wallet**: import, store, manage, present VCs		
		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 context (connection)		
		Delete connection		
		Request access to a context managed by others		
		Grant access to a (local/remote) data context managed by the user		
		Restore agent's data from backup		
		Backup agent's locally held data		
		Sync contexts across user's devices		
	Components	**Browser Extension**		
		Authorization Server: GNAP		
		Replication: auto-merge of commits		
		Messaging: agent-to-agent and agent-to-provider		
		Storage: local context data storage	P	
	APIs	**KERI** DID support		
		Encryption: key pair generation and signing services		
	Platforms	Android		
		iOS		
		Mee Version:	1	2

P = Prototype

v18

Figure 3: Identity agent functionality

- **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 in figure 4.

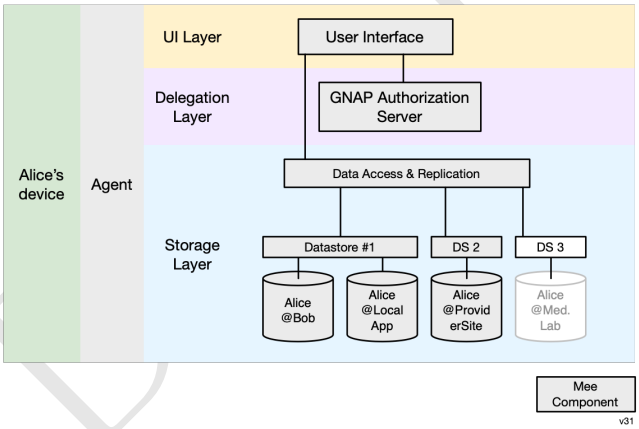

Figure 4: Architecture

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[27] 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.

3.5 Data model

This section describes the data model of an identity agent. The user's data that adheres to this model is replicated across instances of their agent running on different devices, but we focus here on the logical model, not its replicas. At the highest level, the data model can be thought of as a three level hierarchy of data containers (*Container* subclass instances)

[27] oauth.net/gnap/

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 shown in figure 2 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 figure 5 by introducing a new kind of container 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

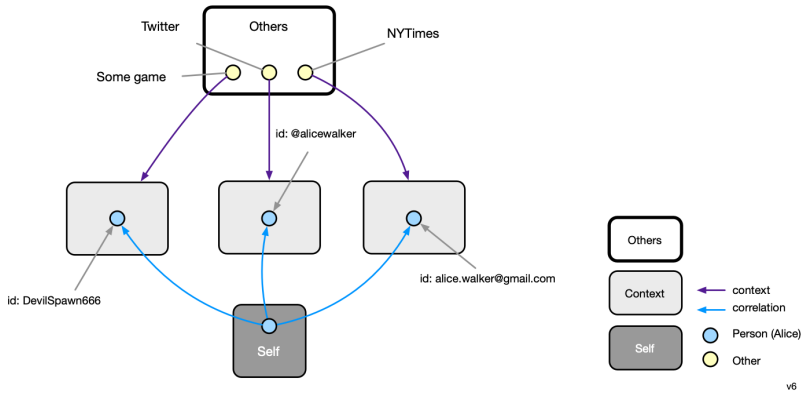

Figure 5: Alices and Others

the data model of the containers themselves. Figure 6 shows the various data container classes.

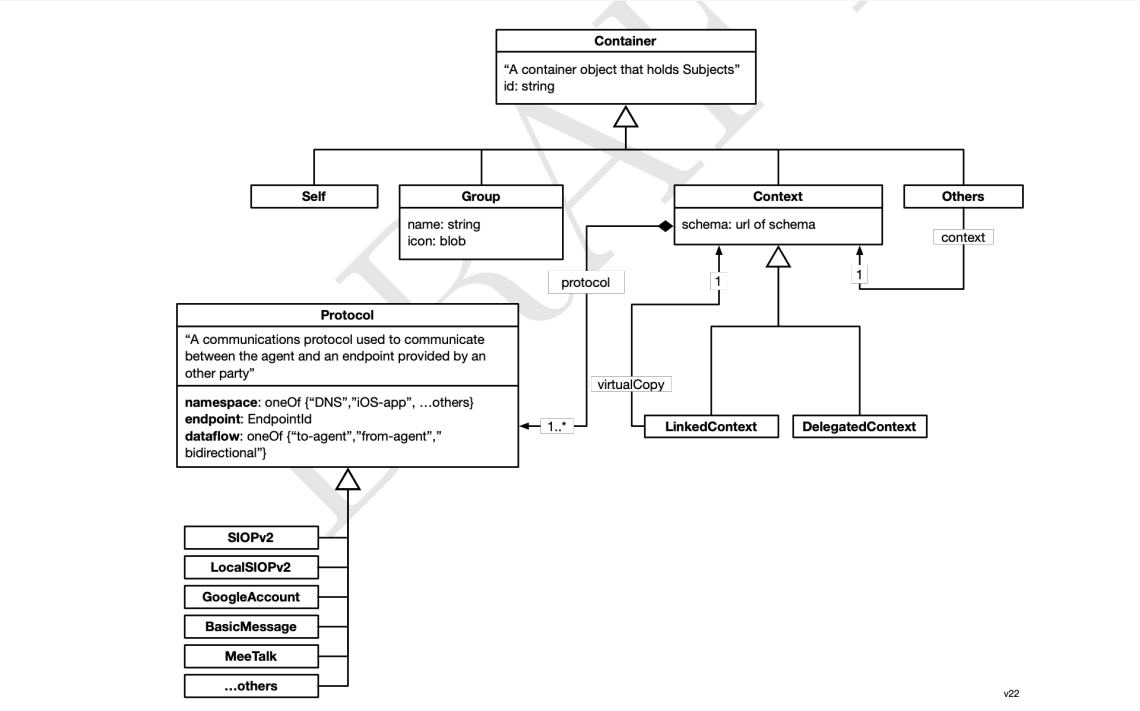

Figure 6: Container classes

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. Figure!7 shows 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.

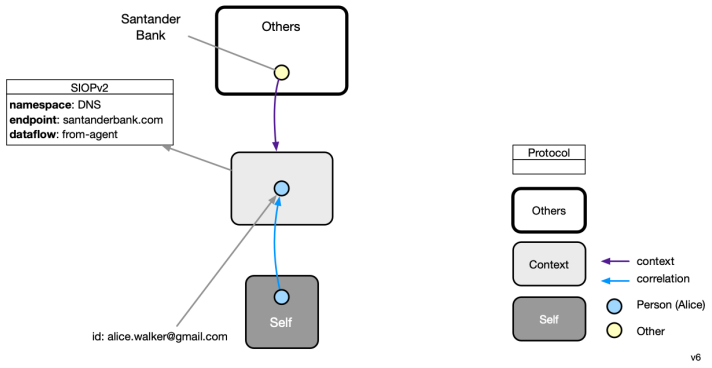

Figure 7: Protocols

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 shown in figure 8, we expand our story about Alice. Now 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 synchro-

nization 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.

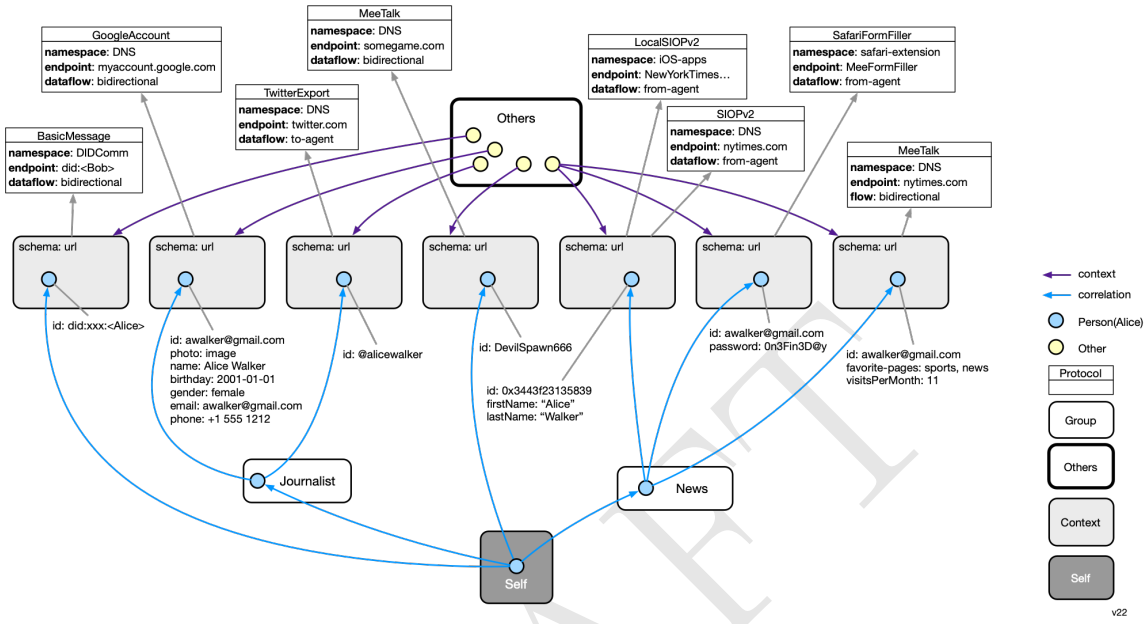

Figure 8: Alice with two groups of connections

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[28] document.

In this example Alice showin in figure 9, 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,

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

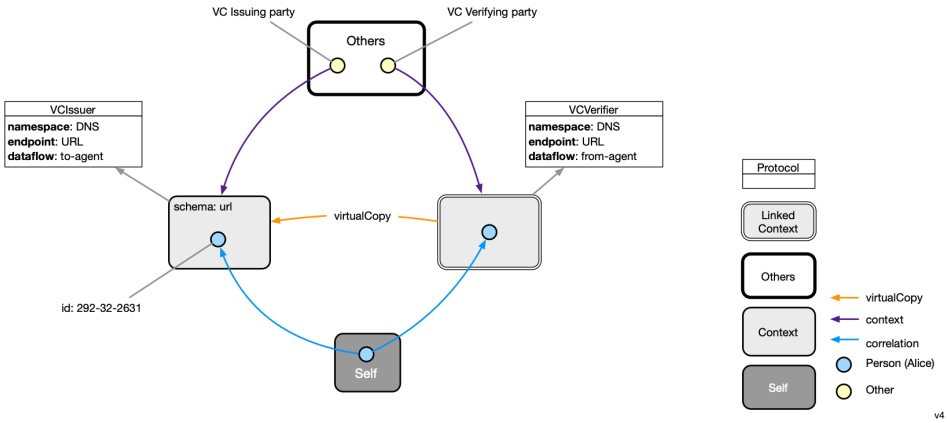

Figure 9: Linked Contexts

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 contain a connection to her bank, the data for which (e.g. her mother's OpenID Connect SIOP claims) is stored in a context within this agent. Using her agent, Alice's mother has delegated access to this context to her daugther Alice.

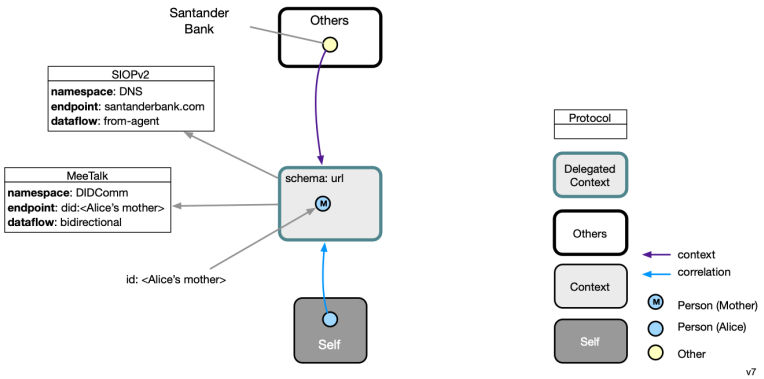

Figure 10: Delegated contexts

As shown in figure 10, Alice has a direct connection with the bank (that communicates via the SIOPv2 protocol) but the context for that connection is linked to the context managed

by Alice's mother's agent. A data replication/sync protocol ("MeeTalk") 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 contain information about subjects (things) that are described according to the *Persona* schema. 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.

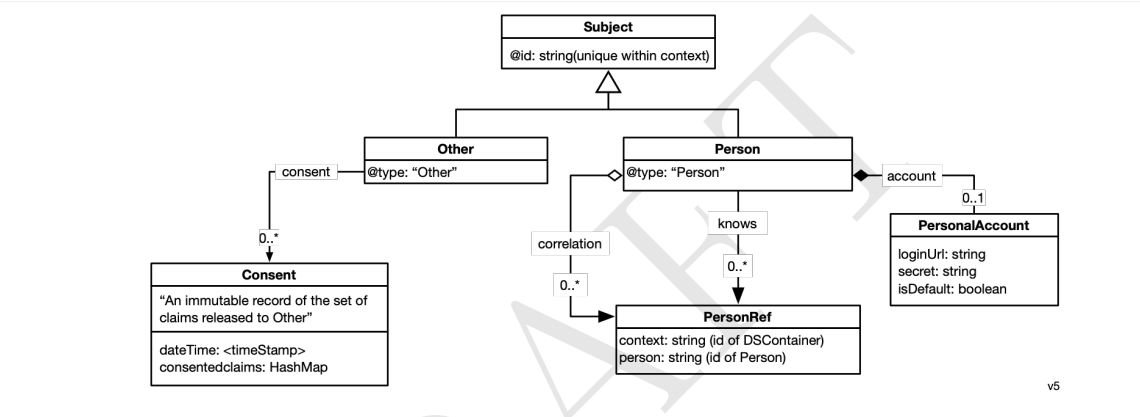

Figure 11: Persona schema

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[29] 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. Some datatype classes are shown in figure 12.

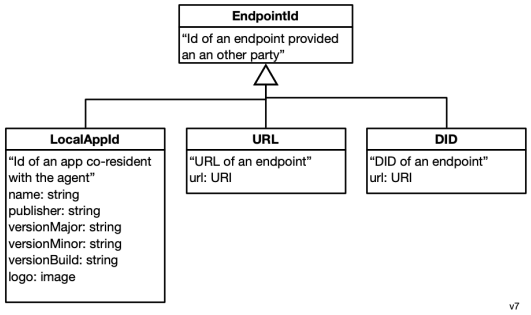

Figure 12: Datatypes

- **EndpointId** - an identifier of an endpoint (e.g. webservice or a local app) supported by an other party.

[29] myaccount.google.com

- **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

We turn now to interactions between an agent and apps.

4.1 Private data sharing

Data held and/or managed by the user's agent is clearly under the user's control. The challenge is that data that the user has shared with another party *also* needs to be under their control. Since no technical means exist to control data held by another party, we rely on law. Current privacy laws and regulations are intended to provide this control, but as we've discussed, place such burdens on the user to effectuate their control that in practice this control hardly exists. The solution we proposed is to combine both legal (license agreement) and technical means (identity agents).

The legal mechanism we propose is a Human Information License (HIL)[30]. The (HIL) is a contract between two parties. The first is the digital service provider legal entity behind a given app. The second is a nonprofit, legal entity called The Mee Foundation (TMF), that represents agent users.

The HIL imposes obligations on 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. It covers information that the user may have shared information manually (e.g. by filling in a form, or other kinds of on-app interactions) or shared with them by a user's agent. The HIL requires the provider to implement *data rights* APIs/protocols that an agent can use to remotely control this collected data. In this way, we tie the legal (HIL) and technical means (agents and APIs) together.

4.2 App-Agent Interactions

In figure 13 we show Alice's agent in the *Agent* layer is interacting with apps in one of two *Application Layers*. The top application layer contains two apps running on a remote

[30] docs.google.com/document/d/13aGk5adoncMxxfl5637NfqP6fl6q_op_1CF50UrJNjg

system. The first is *Bob's Agent* which appears to Alice as an app (just as, by symmetry, Alice's agent appears as an app to Bob's agent). The second is a website labeled *Provider's Website*. The lower application layer contains *local apps* that run on the same device where Alice's device agent is also running. The digram shows a local app labelled *Provider's Local App*.

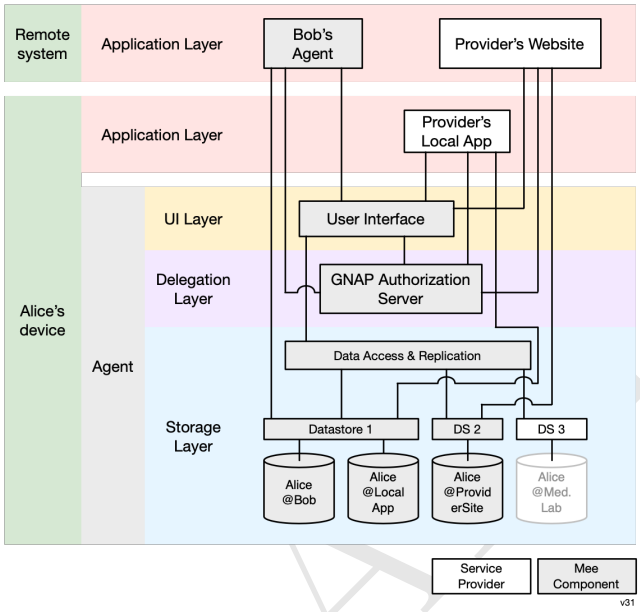

Figure 13: Identity agent interactions with apps

There are four types of intereactions between an app and an agent:

1. **Request** - app needs information from the agent
2. **App-initiated Sync** - app has updated information to sync with the agent
3. **Agent-initiated Sync** - agent has updated information to sync with the app
4. **Delete Context** - user has directed their agent to delete a connection and all of its associated contexts

Request

When a user begins to use an app, there often reaches a point where the app needs information about the user. An example of this is the need for a persistent user identifier. In traditional apps, this request would be a request to create an account or login and is handled using specialized UI ceremonies and specialized authentication flows. In an agent-based architecture, a more general *request* flow is used–a flow that can request an arbitrary

set of *claims* about the user. To initiate this flow the user taps a *Mee* button on the app to initiate a request for information about the user. This tap initiates an OpenID SIOPv2[31] *Authorization Request*.

Figure 14 shows the *request* interaction in a bit more. The agent receives the request messages describing a set of required and optional *claims* (attributes) about the user. The agent searches for relevant attributes/values in its storage layer. The agent then does the some or all of the following:

1. **Discuss with user.** Depending on the search results for one or more claims, the agent and the user may need to discuss what values to return. For example, if two conflicting values are found for the same claim type, the agent may wish to ask the user which (if either) they would like to disclose. If zero or one values are found for a given claim then this step can be skipped, at least for this claim.
2. **Update context.** Populate the context container with the claims (if any) returned from the search.
3. **Display consent screen.** Agent displays consent screen pre-filling what it can and allowing the user to fill in the rest.
4. **Create consent object.** Agent records this consent event.

In the response to the *request* message, the agent returns a non-correlatable *contextId* that can be used for future app-initiated sync operations.

App-Initiated Sync

Apps are required to sync to the agent any changes to claim (aattribute) values, as well as any new attribute values. For example, if the user were to use a webform on the app to update and existing or add a new shipping address, then this information must be synced to the agent. This app-initiated sync operation is shown in figure 15.

Agent-Initiated Sync

When the user is interacting with other apps (i.e. apps other than the current one) a new value of a attribute is generated or captured that perhaps should be updated within the current app context. In this case (given appropriate prior consent by the user) an update value of this claim can be automatically synced from the agent to the app. Figure 16 shows this flow. The agent sends one or more sync messages to the app related to the contextId of the current context.

Context Deletion

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

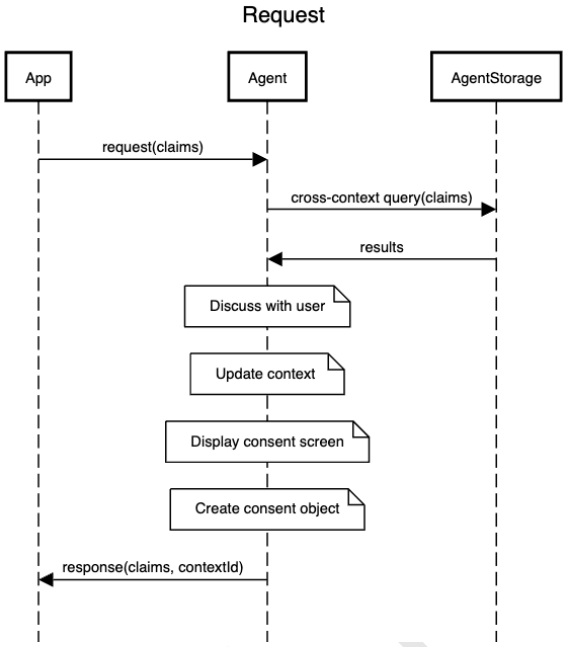

Figure 14: Request

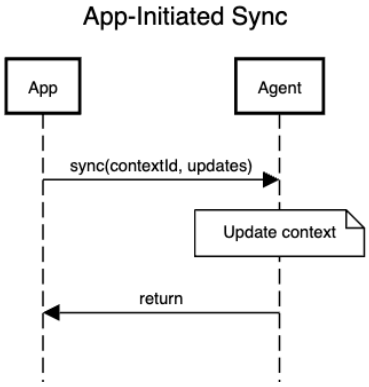

Figure 15: App-Agent Interactions: App-Initiated Sync

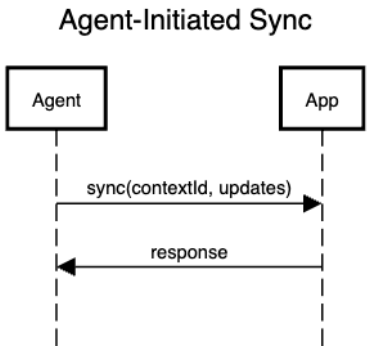

Figure 16: Agent-Initiated Sync

There is one final app-agent interaction, namely, *context deletion*. This occurs whtn the user choses a connection on their agent and taps "Delete." As shown in figure 17 the agent initiates deleteContext messages for each context within the connection.

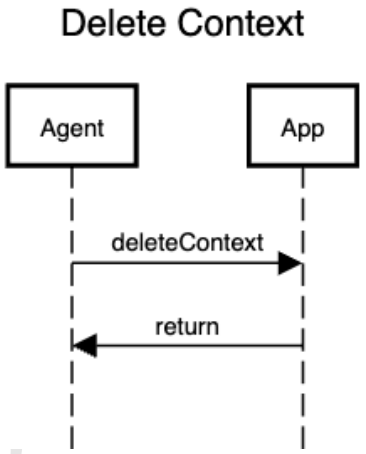

Figure 17: Context Deletion

5 Acknowledgements

Contributors to this paper include Kiril Khalitov, Sergey Kucherenko, Maria Vasuytenko, and Alexander Yuhimenko.

References

[1] Klint Finley. Tim berners-lee, inventor of the web, plots a radical overhaul of his creation — wired. *Wired.com*, 4 2017. URL: `https://www.wired.com/2017/04/tim-berners-lee-inventor-web-plots-radical-overhaul-creation/`.

[2] Adrian Gropper. A human rights approach to personal information technology. *Bill of Health*, 4 2022. URL: `https://blog.petrieflom.law.harvard.edu/2022/04/12/a-human-rights-approach-to-personal-information-technology/`.

[3] 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. URL: `https://jolt.law.harvard.edu/digest/ownership-of-user-held-data-why-property-law-is-the-right-approach`.

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

[5] 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. URL: `https://dl.acm.org/doi/abs/10.1145/3359591.3359737`.

[6] Jaron Lanier. *Who owns the future?* Simon and Schuster, 2014.

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

[8] Eli Pariser. *Eli Pariser: Beware Online" filter Bubbles"*. TED, 2011.

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

[10] Neil Richards. *Why privacy matters*. Oxford University Press, 2021.

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