Identity Agents

Paul Trevithick, The Mee Foundation

March 3, 2023. Revised June 20, 2023

Abstract

There is a power imbalance between digital service providers and the people who use them, with regard to control over people's personal information and online identity. The result is a lack of autonomy and agency. People 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 person 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 the people who use them with regard to control over people's digital identities and personal data.

In the interest of resilience and scalability the internet's original designers 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 people's 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 *personal agent* [9] to refer to software tools that work (i.e. provide agency) "on the individual's side"[11] for, and *exclusively* on behalf of, the person. 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 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[12]. These privacy-eroding symptoms include a lack of personal autonomy, a lack of personal agency, and third-party surveillance.

1.1 Lack of Autonomy

autonomy: *freedom from external control or influence; independence.*[13]

Independence. We each have a "self" that embodies our unique individuality. We "bring" that independent selfness to interactions with others. By contrast, it has been said that online "we have no *digital embodiment.*"[14] 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. We can't "bring" them anywhere. 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*[11] are squarely aimed at addressing this issue–the word "sovereignty" certainly evokes power. Lanier refers to this as a person's right to a "first-class identity."[7, p251].

[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] What Mozilla calls a *user agent*[10]
[11] Project VRM refers to this as "tools for individuals to manage relationships with organizations" to which we would add "...or with other individuals."
[12] Consider the title of Véliz's recent book, "Privacy is Power"[14]
[13] languages.oup.com/google-dictionary-en/
[14] Phil Windley, personal communication, Sepember 2022

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."[15].

As we'll discuss more later on, alternatives have been proposed. One is "user-held" data[4], where your data is held by you in a personal datastore[16]. Another is following "local-first" software principles.[6]

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[17], when people communicate person-to-person online 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 metadata 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

agency: *the capacity, condition, or state of acting or of exerting power* [18]

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, 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.

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

[16] 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`

[17] berty.tech

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

Data presentation. One reason providers' apps rely on form filling and other kinds of data entry is that the person, even one equipped with a personal datastore, lacks the ability to present their 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.

Privacy self-management. Today, despite significant new regulation, the basic approach to protecting privacy hasn't changed since the 1970s. It is often called *notice & consent.* Solove described it using the term *privacy-self management*[13], as follows:

> [T]he law provides people with a set of rights to enable them to make decisions about how to manage their data. These rights consist primarily of rights to notice, access, and consent regarding the collection, use, and disclosure of personal data. The goal of this bundle of rights is to provide people with control over their personal data, and through this control people can decide for themselves how to weigh the costs and benefits of the collection, use, or disclosure of their information.

Although well-intended, and necessary, *notice & consent* does not provide people with meaningful control over their data. The problem is well summarized by [2]:

> When presented with click-through consent, privacy policies or terms of use statements, most people reflexively select "I agree". An extensive body of academic research specifically on privacy and data collection notices demonstrates that members of the public don't read them and might not understand them if they did and that many misinterpret their purpose, assuming that the existence of a privacy policy displayed by way of notice means that the entity collecting the data offers a level of data protection when, in fact, privacy notices do not guarantee privacy. Since the terms offered are typically "take it or leave it", to decline often results in being denied the product or service one seeks, creating a disincentive for consumers to do anything other than accept the terms.

Even if it were cognitively feasible for an interaction with a single entity's app(s), the sheer number of entities collecting and using personal data makes it infeasible for people to manage their privacy separately with each. Privacy-self management doesn't scale from the individual's point of view. They lack agency (i.e. human-centic computational capability) necessary to perform the required tasks.

Privacy policy expression. With a few exceptions, (e.g. the Global Privacy Control[19]), people lack the technical means (i.e. computational capability) to express their own privacy terms to providers.

Data rights. In a growing number of jurisdictions, including Europe under GDPR[20] and California under CPRA[21], the person'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 is sufficiently onerous, that in practice they are not respected. The person must send written requests to get copies of their data, update it, or have it be deleted. The power of personal 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."[10] Pariser's recommendation is that if platforms are going to be gatekeepers, they need to program a sense of civic responsibility 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."[9, p66]

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

1.3 Third-party surveillance

Whereas the person 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 person is unaware, that track and assemble databases about them. Databases of personal data in the hands of hundreds of unknown third-parties creates privacy risks and vulnerabilities. People 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 person's browser).

Surveillance-based targeted advertising. Targeted advertising [22] 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,

[19] globalprivacycontrol.org
[20] gdpr-info.eu/
[21] thecpra.org/
[22] Also known as behavioral advertising, or more recently, interest-based advertising

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 app 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 person's back. They could be considered an unfortunate, privacy-invading *iterim* solution to address the fact that people lack agents with which to directly provide this data about themselves.

1.4 Inconvenience

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

Repetition. When using apps, people 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 database of personal information. The person has the hassle of repeated data entry, and the app offers a less-than-optimal user experience.

Password management. The average person 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 agent, it has been estimated that less than five percent people online use one.

Account management. The person 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 person 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 person not having an agent to manage these relationships.

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 them.

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 person 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 practically impossible.

Tools to manage these chores must sit on the person's side, and work on their behalf across all of them. Technologies of this kind, that empower a person across multiple apps, e.g. browsers and password managers, are called are *user-agents* since they act as agents of the person.

2.2 On-device storage and processing

If we assume a human-centric decentralized architecture, where should the person'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 person's datastore and processing is on their own phones, laptops, and perhaps home servers. Cloud-based means that the person's datastore and processing lives primarily in the cloud (e.g. on a SOLID[23] pod). We say *primarily* because there are usually use-cases that involved replicating/syncing some of the data to the "other" location. The local-first software[24] principle are highly relevant.

Security. Although this is debatable, it is our contention that given a large number of people, having a personal datastore on-device is more secure than in the cloud. Even if each alternative were equivalently secure for a single person, 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. Any 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 people 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.

[23] solidproject.org

[24] www.inkandswitch.com/local-first/

2.3 Replication

If we assume most of a person's information is held on-device, we need to solve the roaming problem when a person has two or more agents running on multiple devices each of which is only intermittently connected to the internet. The person's data needs to be kept consistent across these agents and devices, at least eventually. This requires that the person's agents implement data replication and syncing between themselves in a peer-to-peer (P2P) fashion. Unfortunately pure P2P communication between agents running on differing device platforms remains an unsolved problem and intermediate relay servers are required.

Since relays are a necessary part of the deployment architecture, for privacy, autonomy, trust, and security reasons they are subject to their own design considerations. We touch on a few of them here. For the very few people who are able and willing to self-host their own relays, the relay needs to be free, open-source and easy to build and deploy. Everyone else will have to trust some external administrative authority. Hopefully relays will be available freely or at very low cost. In either the self-hosted or external case, the relay needs to be trusted. For this reason its source code should be open and the relay should only store encrypted data. It should store message data only while waiting for the recipient agent to come online.

2.4 Backup

One disadvantage of *non-custodial*, on-device architectures (as compared to cloud-based architectures) is the vulnerability agent owners who are not diligent about backing up their devices (e.g. to an online service) face of losing their agent-managed personal data. For people with more than one device this is less likely since data is replicated (as mentioned above) across their devices, and a repaired or replaced device's data can be restored from one of the person's other devices. There remains of course the worst-case scenario wherein the person hasn't backed up any of their devices and all of them are lost or damaged simultaneously.

Agents could implement backup/restore approaches that would provide recovery from even this disaster, however they are themselves complex and problematic. The agent's data must stored in remote storage location(s) and encrypted using a master passphrase that the person must never be able to forget or lose. To do this, various approaches including sharding, shared secrets[25], and social recovery have been proposed but this is still an emerging area.

[25] en.wikipedia.org/wiki/Shamir%27s_secret_sharing

2.5 Loyalty

Much of the power asymmetry described in the first section is due to economic incentives for online service providers. These incentives motivate them do just enough in the person's interest to keep them using the service and generating personal data that the provider can monetize. Personal data, after all, is now considered to be an asset class–the more of it that's collected the better. For an agent to work *exclusively* on behalf of the person, the *agent provider* (i.e. organization that develops and publishes it) must not have an economic incentive to be disloyal to the person's interests. One way to ensure this is to design the agent such that its data remains within itself and is never stored by, or even accessible by, the agent provider.

2.6 Metacontextuality

Zuckerberg once said that "[h]aving two identities for yourself is an example of a lack of integrity"[5]. 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 person 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."[12, 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 recommendation 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[26] Kim Cameron[27] introduced two useful definitions based on archaic English:

[26] www.youtube.com/watch?v=9DExNTY3QAk

[27] en.wikipedia.org/wiki/Kim_Cameron_(computer_scientist)

- **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 embrance and explicitly support the complicated, multi-contextual nature of our lives.

2.7 Delegation

In *A Human Rights Approach to Personal Information Technology*[3] 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[28] at the 2022 Identiverse conference provides additional detail (see slides[29]). 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.

2.8 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–people must have confidence that their information isn't being used against their interests.

The transparency of open-source software can to help build confidence that the technology 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 person's interest.

In addition to open-source, people will also consider the nature of the organization offering the solution as to trustworthiness. The organization's financial incentives should be aligned with their member's interests. A nonprofit organization could be formed which has no financial or business incentive to exploit their member's data against that member's interest. Ideally the organization would not need to have any access to personal data and thus no need to have to trust them, their security infrastructure, their processes, etc.

2.9 Data Governance

Once data is shared from the agent to a first-party there are no technical means to constrain what the recipient can do with it. No technical means, for example, can prevent them from selling it others. Instead, legal means must be employed. Existing privacy regulation is insufficient, so we propose that first-parties sign a Human Information License (HIL) to license the person's information. The HIL terms are fair and balanced, and respect the

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

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

person's privacy rights. This contract is signed by a trusted organization[30] that represents the community of identity agent owners thereby making the processes effortless for them. Lasly, this organization is responsible for enforcement of the contract's terms, again, on behalf of the individual.

2.10 Data Rights

In many jurisdictions people have *data rights* to access, correct and delete their own personal information managed by providers. Privacy regulations state these rights, but in practice the burden of exercising them across hundreds of apps is unmanageable. People need agents that can automate these processes, and thereby reduce the amount of work to a practical level.

3 Identity Agents

We now propose a solution to the problems described in the first section that is consistent with the design considerations in the second section. Our solution, which we call an *Identity Agent*, is a trusted, personal agent[31] that manages a person's online identity and personal information.

3.1 The human perspective

An *identity agent* gives an individual control over their personal information as they interact with websites, mobile apps, and other people's agents. It does this through a combination of technical and legal mechanisms.

3.1.1 Privacy and Autonomy

The agent is installed as an app and runs on a person's "edge" devices (e.g., mobile phone, laptop, etc.) where, entirely under their control, it maintains a local, private database of their personal information. By default none of this information is shared with any other entity, including the organization that develops the agent and provides ancilliary services to support it. When an app wants to know something about the person, the agent shares as much or as little as the person chooses. If the app provider signs a Human Information License and thereby becomes *certified*, they are thereby obligated to (i) require explicit consent for collection, processing, storage and sharing of the person's data and to (ii) implement APIs to exercise the person's right to access, correct and delete their personal information. Agents could also provide ad profiles to certified publisher websites that

[30] These kinds of organizations have been variously described in the literature as "data unions," "data coalitions," "Mediators of Individual Data" (MIDs) by Lanier et al.[8], etc.

[31] Similar ideas have been proposed by others. See *personal user agents*, in [2, p24]

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. Working solely in the person's interest, it collects information from apps that already hold their data and shares it with other apps that need it. Our vision is that that the person *never has to repeat themself* (nor remember passwords!) as they move from app to app across the internet.

Here are a few examples. If an app wanted to know the person's email address, it might ask for it in a web form. In this case the agent would use its form-filler "protocol" to fill in the value. If the app supports password-less sign-in (e.g. using OpenID Connect) the agent acts as the identity provider. If an app 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 app. 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 agent be human-centric and put the person at the center of all of their online relationships.

3.2 Self and Contexts

The agent represents both the person's single *selfness* and their multiple context-dependent *whonesses* that they have in their interactions with other apps and people.

The selfness of the person 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 person'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 person in each context. To ensure privacy only the person 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 person might play a game on a gaming app using the id DevilSpawn666, while communicating on Twitter as @alicewalker and subscribing to the Olde York Times as alice.walker@gmail.com. Figure 2 shows a simplified view of how this is represented:

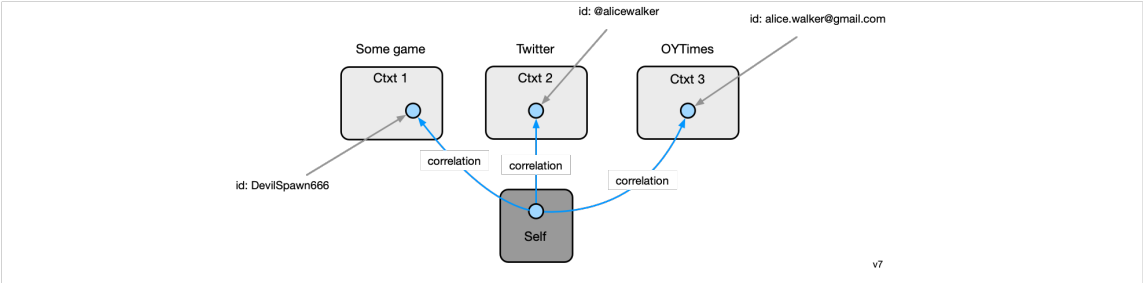

Figure 2: Alice with three contexts

3.3 Functionality

Figure 3 shows a summary of the functionality of an identity agent.

Identity Agent	**Connectors**	SD-JWT-based VC presentation		
		SD-JWT-based VC issuance		
		Google Account		
		Global Privacy Control		
		OpenID SIOPv2 (used by Connect-with-Mee)		
	Functions	**Chat:** Person-to-person and agent-to-person messaging		
		Request access to a context managed by others		
		Grant access to a (local/remote) data context managed by the user		
		Sync contexts across user's devices		
		Delete connection		
		Consent to share required/optional data with a service provider		
		Edit data in self-asserted contexts		
		View data in context (connection)		
		Recognize user (e.g. using facial recognition, etc.)		
	Platforms	**Browser Extension** (e.g. Mobile Safari)	P	
		Android		
		iOS		
		Mee Version:	1	1.1

v19 P = Prototype

Figure 3: Identity agent functionality

3.3.1 Connectors

An identity agent is a tool to allow its user to manage data *connections* with apps and agents of other people. Due to the different communication protocols and data storage approaches involved in these connections, agents use an extensible architecture that leverages a set of *connectors*.

Here are a few examples of connectors:

- SD JWT-based VC presentation - present Verifiable Credentials from the agent (wallet)
- SD JWT-based VC issuance - store a Verifiable Credential in the agent (wallet)
- Gooble Account - pull data from myaccount.google.com
- Global Privacy Control - sends a "Do Not Sell My Personal Information" signal to apps
- OpenID SIOPv2 - allows the person to authenticate with an app without using passwords, without first creating an account, and with surveillance by an external identity provider.

3.3.2 Functionality

Here are functions exposed to the person through the UI:

- **Chat:** Person-to-person and agent-to-person messaging
- **Request** access to another person's information
- **Grant** access to selected portions of your information to another person
- **Sync** contexts across person's devices
- **Delete connection** delete all data associated with this set of contexts
- **Consent** to share required/optional data with a service provider
- **Edit** data in self-asserted contexts within a connection
- **View** data related to a connection
- **Recognize** the agent ownder (e.g. using facial recognition, etc.) and thereby prevent others from using the agent

3.4 Architecture

In this section we present the architecture of an identity agent. The multi-layered architecture of Alice's agent is shown in the center of Figure 4. We concentrate here on the agent itself and leave a discussion of its interactions with the four apps (one to the left and three to the right of the agent) to section 4.2 later on.

This section references terms such as *self*, *context*, *connection*, and *protocol* that are desribed in section 3.5 where we describe the agent's data model.

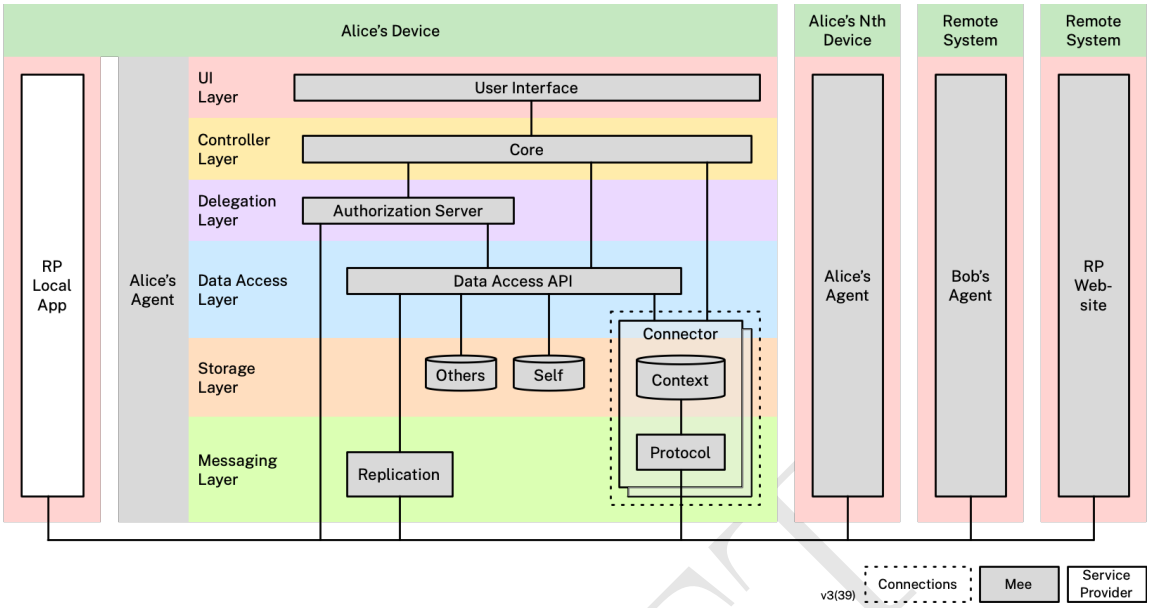

Figure 4: Identity Agent Architecture

3.4.1 User Interface layer

Alice's agent is deployed on one or more of her devices (e.g. a smart phone, laptop, etc.). The UI layer provides her with a UI that she can use to manage her data sharing relationships with service providers (aka relying parties) and other agent owners. Using this UI she can add and delete connections. Within each connection she can consent to data shared from her agent, see what data is involved in the connection, and in some cases edit attribute values.

3.4.2 Controller layer

The controller layer manages requests from the UI using the services from the layers below it. It manages the person's interactions with the *authorization server*. It updates context attributes via the *data access API* abstraction based on commands from the UI. Lastly, it is responsible for creating and deleting *connections* and their component *connector(s)*.

3.4.3 Delegation Layer

The *authorization server*[32] (AS) responds to requests for access to Alice's contexts–contexts which are manaaged by the data access Layer described below. The AS sends these requests to the controller layer (which may in turn call back to the UI Layer) to allow the person,

[32]For example, a GNAP `oauth.net/gnap/` Authorization Server

either interactively or by policy, to grant or deny them. These requests can come from other people's agents and other entities that support the agent's delegation protocol.

3.4.4 Data Access Layer

The data access layer contains the *data access API*. This API provides a common interface to personal data across both context datastores managed directly by the agent, as well as context datastores managed by *connectors*. In either case it provides an interface to the personal data contained in the contexts described in section 3.5. This layer is also responsible for replicating all agent state to other agents that the person owns. It does so by sharing updates to other agents via the messaging layer.

3.4.5 Storage Layer

The storage layer contains three kinds of datastores. The first is a local store called *others* that stores Alice's representations of other parties with which Alice interacts. The second is used to store the person's *self*[33]. The third are context datastores managed by connectors internally and thus hidden from the agent core.

3.4.6 Messaging Layer

The messaging layer consists of a set of libraries for specfic communications protocols. Two kinds of protocols are shown in Figure 4. The first kind of these is the built-in Replication protocol used to replicate Alice's contexts across her devices. The second kind are the protocols implemented by connectors to communicate with external relying parties (e.g. apps or other people's agents) or other parties that have delegated to Alice access to one or more of their contexts. This second kind of protocol is described in the *Protocols* subsection in section 3.5.

3.4.7 Connectors

Now that we've described the horizontal layers of an agent, we turn to the *connector* extension point. A single agent has multiple *connections* each of which is implemented by one or more multiple connectors. each of which implements a specific communications protocol and stores data related to a single context of a connection with another party.

The connector interface hides the details of its storage and its communications from the outer agent framework. As you can see in Figure 4 connectors span the storage and messaging layers. A connector implements the data access API but is free to store its context's data in any schema and using any storage technology it chooses. It also communicates using whatever communications protocol is need to share its data with the relying party.

[33]See section 3.5 for definitions of *others* and *self*

3.5 Data model

This section describes the data model of an identity agent. The person's data adhereing to this model is replicated across two or more agents 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) 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 person as a single individual. The Self has a set of Context containers each of which represents how the person is presented to or perceived by another party (e.g. another person's agent or a digital service provider's app)–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 person) 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 agent owner. 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 person might play in a set of child contexts.

In the simplified example shown in Figure 2 a person, 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 Olde 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 of the other party). We have added three of these other parties explicitly to Figure 5, and added a new kind of container, called Others, to contain them.

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 person directly into the agent. Or it may be claims entered by the person on an app 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 person.

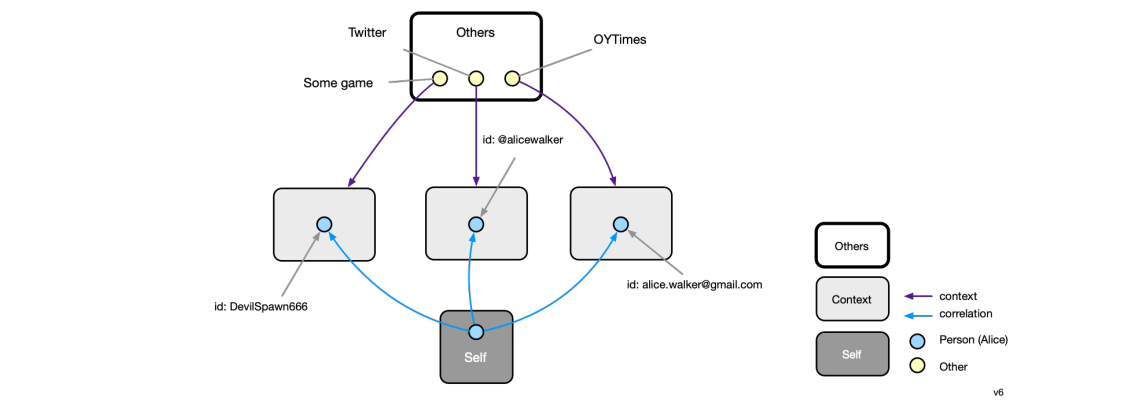

Figure 5: Alice's Self and Others

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 held by these containers. Let us start describing the data model of the containers themselves. Figure 6 shows 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 person 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 person
- **Group** - an intermediate level container that holds a single Person node that represents a common role or persona that the person 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 person in a specific aspect of their relationship with some other party. We say "specific aspect" because the relationship between the person a given other, may be represented by more than one context, each representing a different aspect.

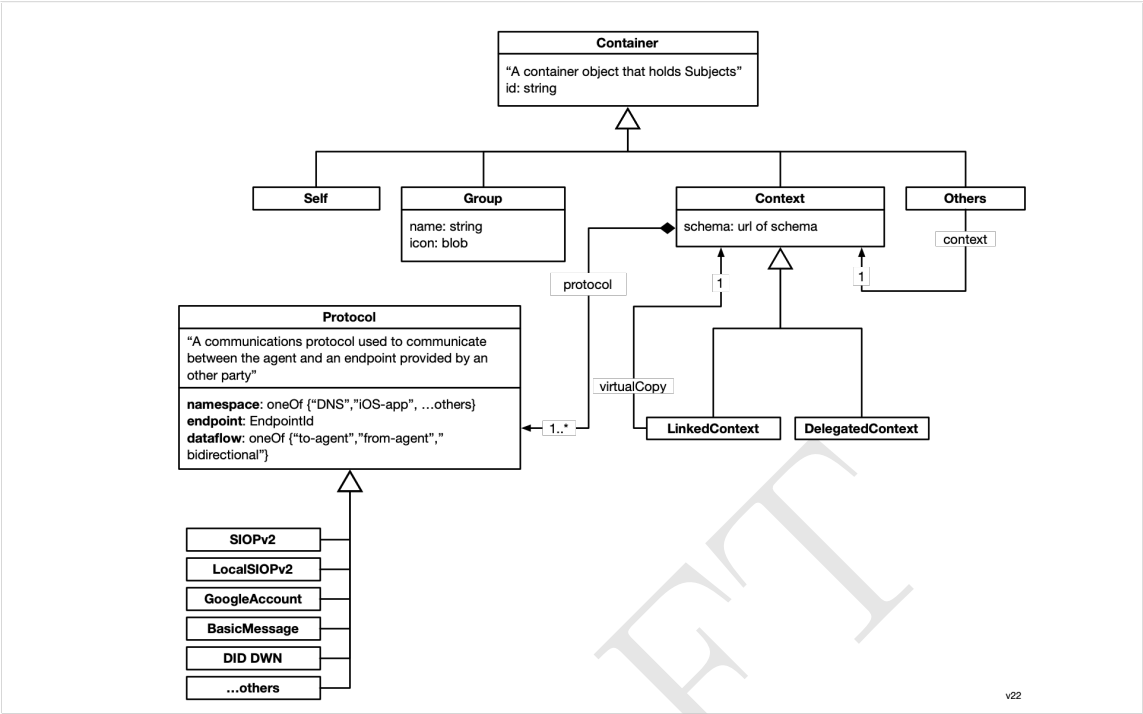

Figure 6: Container classes

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 commu-

nications 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 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.

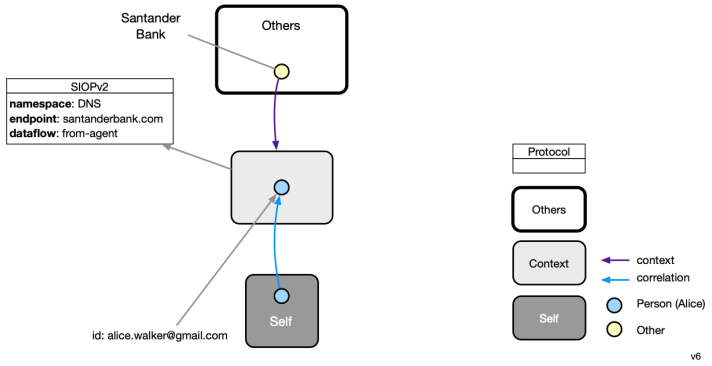

Figure 7: Protocols

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 person 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. Alice has organized her connections into two groups. The first represents her role as a journalist, and it contains two connections. The first connection contains a context representing her relationship with Google. 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).The second connection contains a context representing her relationship with Twitter. Her Twitter context contains a snapshot of all of her Twitter account information, lists of who she follows, etc.

Her second group, entitled "News" contains one connection comprised of a Person linked to three contexts, all of which are associated with various facets of her relationship to the Olde York Times. The first of these three is the context that she uses, via SIOPv2 to login to the OYTimes 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 OYTimes using a new (and 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.

Alice also has two free standing connections (i.e. connections that are not part of any group). At the far left is her connection to the agent of another person, Bob. In the middle is a connection representing her relationship with a game she likes to play. This connection is also comprised of a single context.

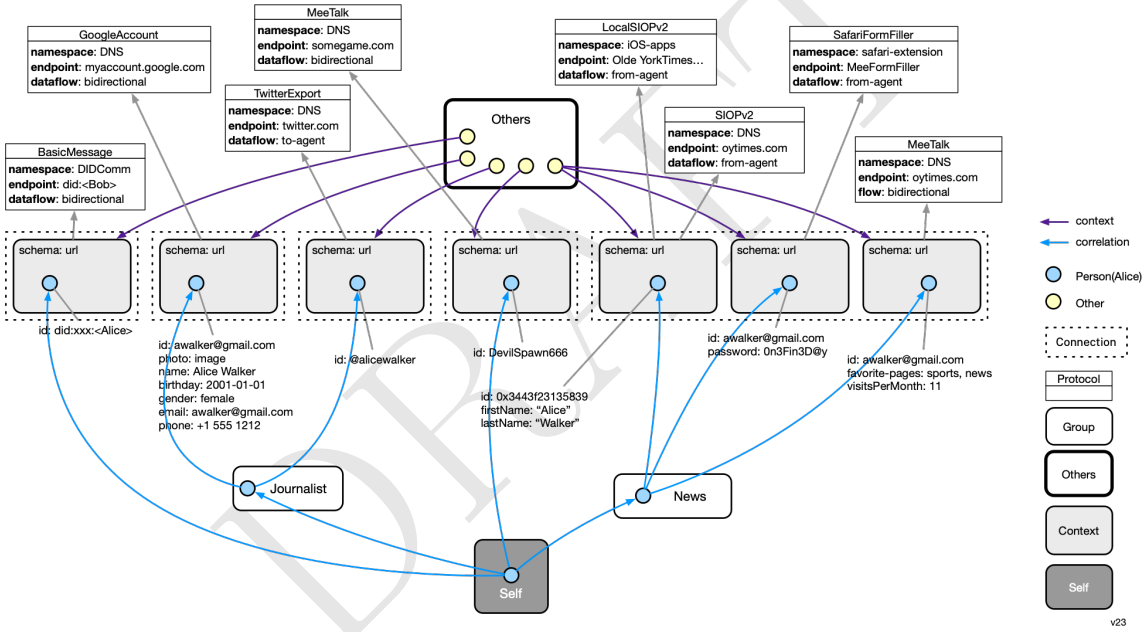

Figure 8: Alice's five connections

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[34] document.

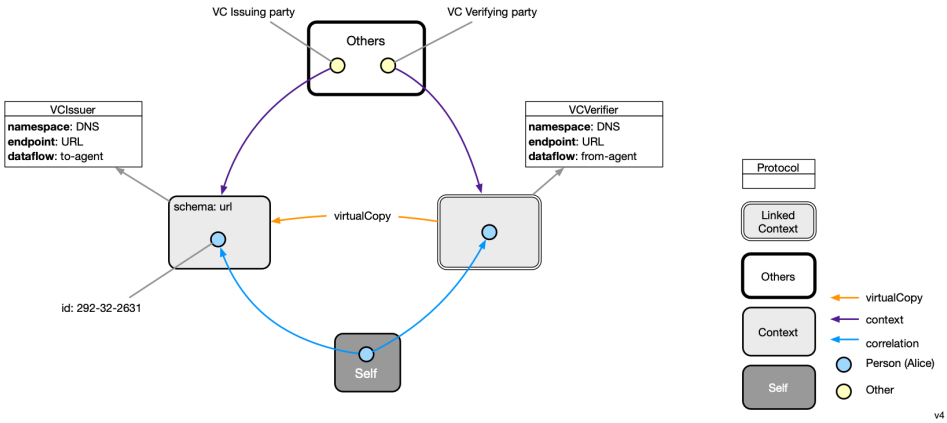

Figure 9: Linked Contexts

In this example Alice shown in in Figure 9, presumably having authenticated herself in her connection to an 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 app of the "VC Verifying party" and finds that they trust the issuing party and thus 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 containing a connection to her bank, the data for which (e.g. her mother's OpenID Connect SIOP claims) is stored in one of the contexts representing this connection. Using her agent, Alice's mother has delegated access to this context to her daughter Alice.

As shown in Figure 10, Alice's mother's connection with her bank is represented by a delegated context. Alice now has the ability to view (and potentially update) information in this context. Information about Alice's mother's account information at the bank might be helpful for Alice to have while taking care of her mother. Data replication/synchronization is used to ensure that Alice's DelegatedContext is always synchronized with the "original" context on her mother's agent.

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

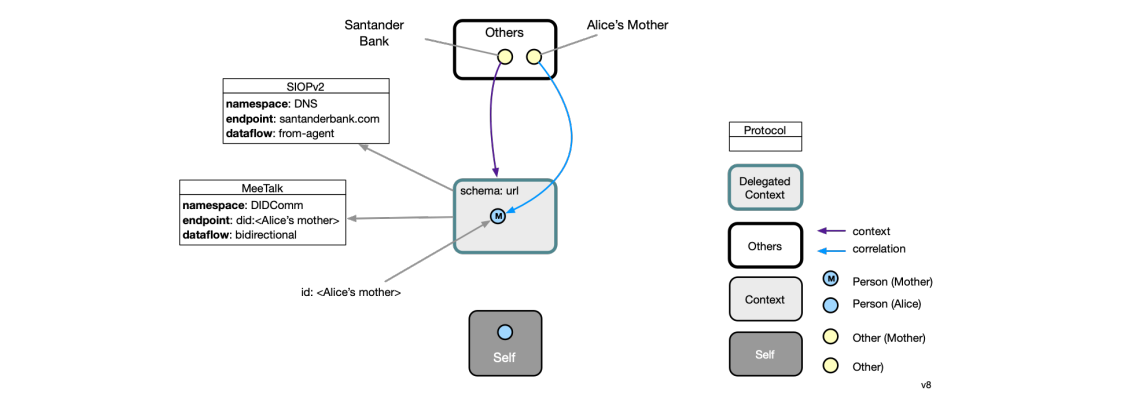

Figure 10: Alice's agent with a connection using a delegated context

Whereas the main point here is giving Alice visibility into her mother's bank, information, it may be possible in some cases for Alice's agent to use this information to authenticate as her mother to the bank (although authenticating as someone else (especially in the case of abank) is usually a violation of the terms of service of the other party.

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.

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

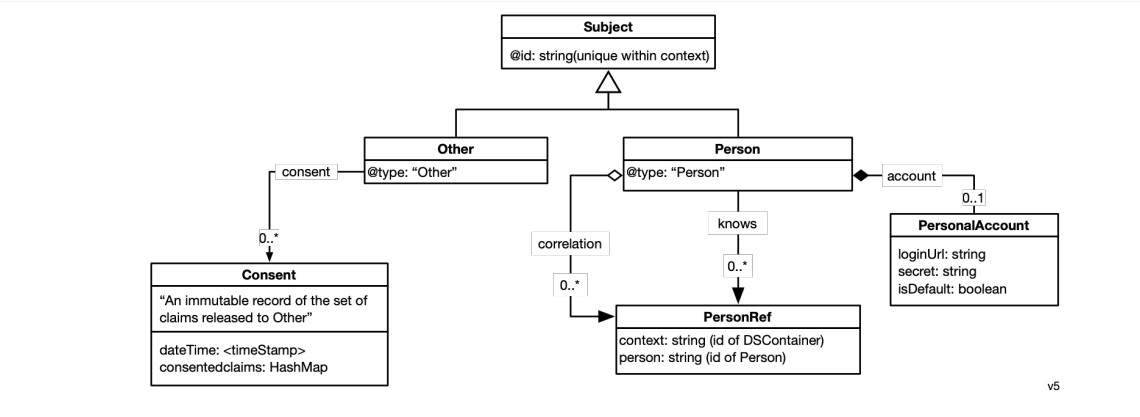

Figure 11: Persona schema

  - **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 agent owner)

- **Other** - a Subject representing another person or a legal entity with which the agent owner has a connection. Each Other object has:
  - **consents** - zero or more Consent objects. Each Consent has:
    * **dateTime** - time stamp of when the person 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[35] 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.

[35] myaccount.google.com

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.

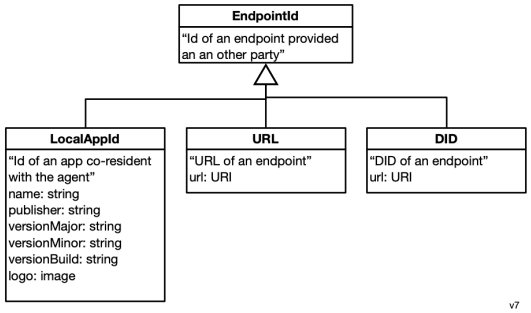

Figure 12: Datatypes

- **EndpointId** - an identifier of an endpoint (e.g. webservice or a local app) supported by an other party.
- **LocalAppId** - A specific kind of EndpointId. Uniquely identifies a service provider's mobile app.

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 person's agent and stored on-device, is inherently under the person's control. Data that the person shares with another party or is collected by them in other ways *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 person 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 the Human Information License (HIL)[36]. The (HIL) is a contract between two parties. The first is the digital service provider. The second is a

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

nonprofit, organization called The Mee Foundation (TMF), that represents the community of agent owners. The TMF is a *Mediator of Individual Data* (MID), a term coined by Lanier et al.[8], that enforces the terms of the HIL on behalf of the community.

The HIL imposes obligations on the provider. Among them is the provider's requirement to respect the person's *data rights* to access, correction (editing), and deletion of the information collected and held by them. It covers information that the person 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 person's agent. The HIL requires the provider to implement *data rights* APIs that an agent uses to remotely control this app-held data. In this way, we tie the legal (HIL) and technical means (agents and APIs) together.

The HIL's provisions are intentionally generic. They are designed to meet the needs of the entire community of agent owners. We expect that other contracts containing more specific provisions will be required to meet the needs of more specialized communities. Each community can amend the HIL to meet the specifics they require, provided that they do not weaken the HIL's existing provisions and protections. These specialized communities would organize, govern and operate independent MIDs that enforce their more specialized HIL-based contracts. These specialized MIDs would enter into agreements with one or more providers which would be held to both the generic terms of the HIL as well as the additional, specialized terms.

4.2 App-Agent Interactions

In Figure 13 (a repeat of Figure 4) we show Alice's agent interacting with four apps. At the far left we is an RP Local App running on the same device. On the right we show three other apps. The first is another instance of Alice's agent running on another of her devices. The second is Bob's agent communicating with Alice's agent using the replication protocol common to both agents. The third is an RP's website.

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

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

Request

When a person uses an app, during the interaction the app may need information about them. The information requested is usually about the person, but could be about other

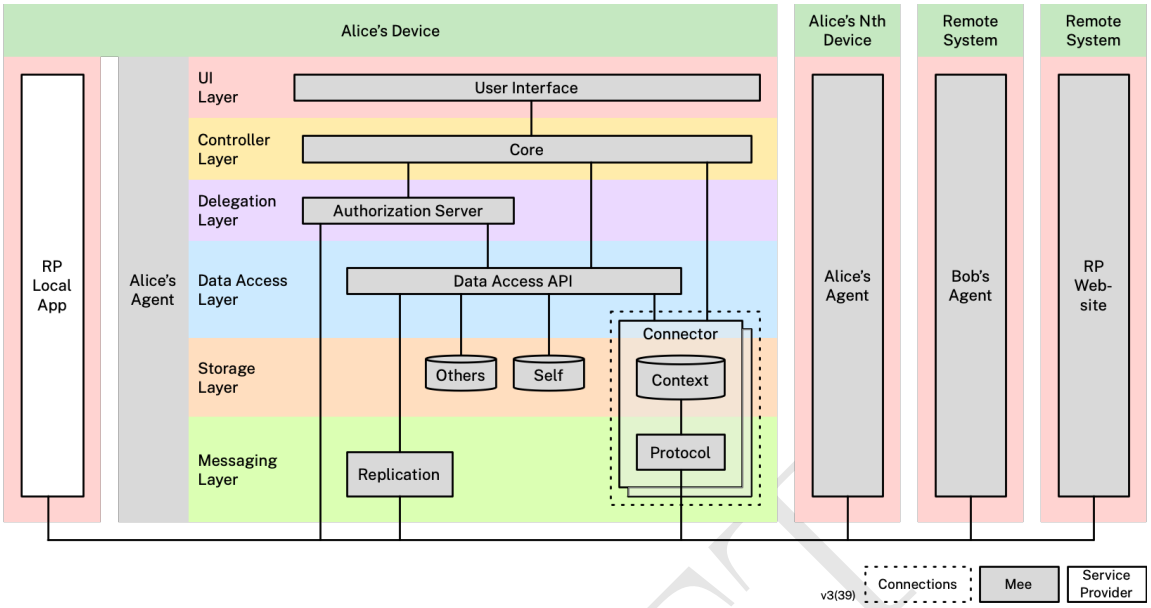

Figure 13: Identity Agent Architecture

people or anything else. The information requested may be a simple list of attributes and values, or something more complicated. The app can express whether each of the requested attributes is required or optional.

Authentication is request wherein the app wants to know the identity of the person so it can know if this a first-time or a returning person. Traditionally, an authentication request is implemented using a login/sign-in interaction. However in an agent-based architecture, a *request* message can be used. To initiate a request the person taps a *Connect-with-Mee* button on the app. This tap initiates an OpenID SIOPv2[37] *Authorization Request*. In the narrow context of authentication the information requested and returned are often called *claims* since they are usually claims made by some entity (possibly the person) about the person. If additional information is needed at any point in the session, the app can again display the Connect-with-Mee button.

Figure 14 shows the *request* interaction in a bit more detail. The agent receives the request message which contains a query describing the kind of information required and/or desired. The agent searches for relevant objects and/or attributes in its storage layer. The agent then does the some or all of the following:

1. **Discuss with person.** Depending on the search results for the information, the agent and the person may need to discuss what object and/or attribute values to

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

return. For example, if two conflicting values are found for the same attribute type, the agent may wish to ask the person 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 person 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 person 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 person 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 person) 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

There is one final app-agent interaction, namely, *context deletion*. This occurs when 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.

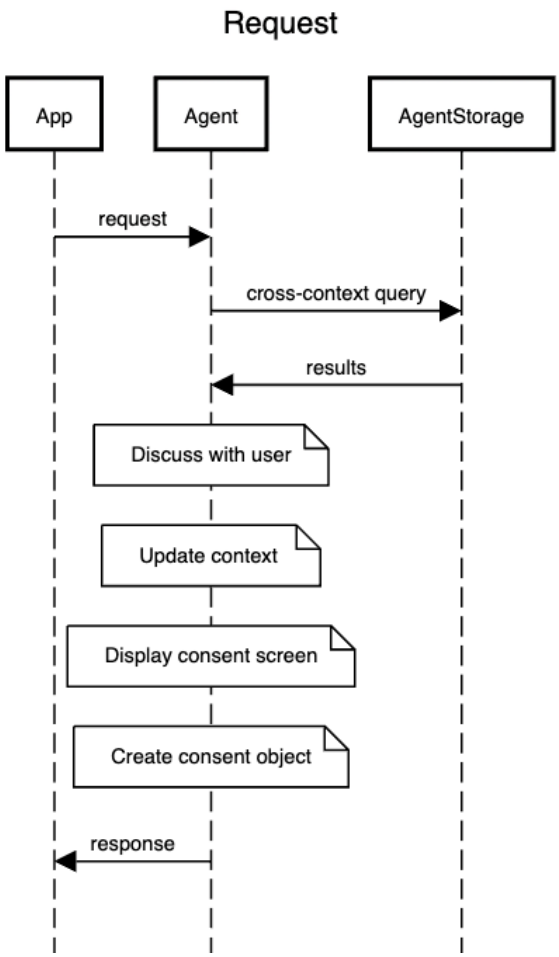

Figure 14: Request

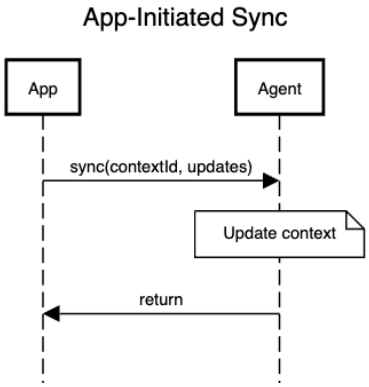

Figure 15: App-Agent Interactions: App-Initiated Sync

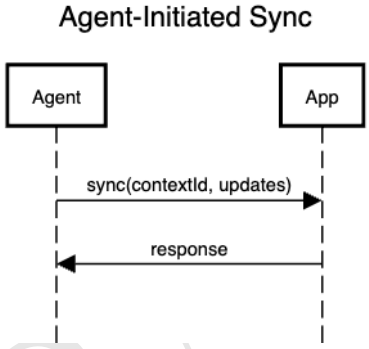

Figure 16: Agent-Initiated Sync

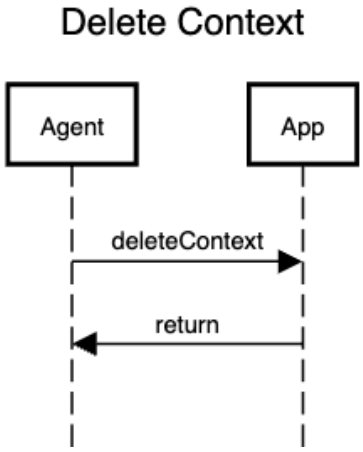

Figure 17: Context Deletion

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] Anne Josephine Flanagan, Jen King, and Sheila Warren. Redesigning data privacy: Reimagining notice & consent for human technology interaction. *World Economic Forum*, 2020. URL: `https://www.weforum.org/reports/redesigning-data-privacy-reimagining-notice-consent-for-humantechnology-interaction`.

[3] 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/`.

[4] 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`.

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

[6] 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`.

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

[8] Jaron Lanier and E Glen Weyl. A blueprint for a better digital society. *Harvard Business Review*, 26, 2018. URL: `http://eliassi.org/lanier_and_weyl_hbr2018.pdf`.

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

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

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

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

[13] Daniel J Solove. Introduction: Privacy self-management and the consent dilemma. *Harv. L. Rev.*, 126:1880, 2012. URL: https://scholarship.law.gwu.edu/cgi/viewcontent.cgi?article=2093&context=faculty_publications.

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