UVCID Identity and Data Interoperability Paper

HAIKU, MOTHER, UVCID, BABEL: Technical Implementation

Paul Hodgson, Indika Piyasena, Shawn McLean, Farshad Niayesh, Maziar Sadri

Abstract—**There is currently no unified method for standardisation and transfer of data via the blockchain. The result of this is a fragmented data industry, in which enterprises cannot transfer data in a seamless fashion. The solution for this is Unification's HAIKU protocol, which deploys a series of Software Development Kits (SDKs) as CAPSULEs on enterprise servers to standardise and encrypt data. This allows actors to transfer data peer-to-peer outside the blockchain via state channels while the low-cost, on-chain transaction is a hash that is used to validate the data and assign a possible payment in UND via a smart contract. Further, Unification's Unified Verifiable Credential ID (UVCID) infrastructure will allow actors to securely issue, hold, distribute and cryptographically verify personal information. The result is an ecosystem of data transference in which B2B, C2B and intranets can interact with each other in a more efficient manner.**

I. Introduction

Unification is a blockchain-agnostic ecosystem of Data Providers, Data Consumers and End Users that utilises blockchain technology, machine learning and auxiliary software components to allow the monetised, secure transfer of data.

The ecosystem is built around a web of smart contracts, which allow the initialisation and flow of data transfer agreements between Data Providers and Data Consumers, while putting the control of the data flow in the hands of the users themselves, who ultimately own - and are remunerated for - their data.

The underlying blockchain provides security and transparency for the users that have granted or denied access to their data. It also provides the infrastructure to remunerate actors in the ecosystem. The supporting client and server software[1] track the on-chain state of a data transfer at various stages, and channels the off-chain processes involved with request validation, data extraction, transformation and the secure transfer of the standardised data. Additionally, the server software ensures that data from diverse sources remains unified at the user level.

Unification maintains a curated repository of UApps within the ecosystem, which are discoverable to Data Consumers and End Users via the BABEL Front End app.

II. Actors and Goals

The main actors of the Unification system are the Data Providers, Data Consumers and the End Users, while the Unified Verifiable Credential ID system introduces the concept of Issuers, Holders and Verifiers.

Data Providers are traditionally data silos. These players collect user data by providing engaging applications, and before Unification, they have rarely been incentivised sufficiently to share this data. A Data Provider can publish their data for revenue by installing an application developed by the Unification Foundation called HAIKU. Examples of Data Providers are Fitbit and Garmin.

Data Consumers can be Data Providers (as described above) who would like to extend their data set, or independent entities who would like to access data (for example, to undertake research). In either case, Data Consumers are actors willing to pay for data. An example of a Data Consumer is Stanford University.

End Users are the application users (e.g. users of Fitbit or Garmin) that have been providing data to those applications, through their interactions with them. End Users in the Unification ecosystem are incentivised to openly share their data for remuneration and the potential for being presented with better products and services. Alice and Bob are examples of End Users.

Issuers are entities who generate and sign Verifiable Credentials, and issue them to Holders. Issuers may also be Data Providers and Data Consumers within Unification, or standalone entities.

Verifiers are entities who receive Verifiable Credentials, typically from Holders, and cryptographically verify and use the contents of the credentials. Verifiers may also be Data Providers and Data Consumers, or standalone entities.

Holders are entities who store and distribute Verifiable Credentials which have been issued to them. Holders are typically, but not limited to, End Users within Unification.

A. *Use Case Synopsis*

Unification's decoupled architecture toolkits combine machine learning and efficient, transparent data transfers. This allows the development of autonomous applications designed to revolutionise enterprise and improve our everyday lives.

Some of the use cases where these tools optimise business processes include:

- Building more efficient data transfers via state channels internally among different business divisions to optimise partnerships and improve user product experience.
- Building more efficient data transfers via state channels externally with partners to optimise partnerships and improve user product experience.

[1] Code references and GitHub links to the Unification software used within this paper are for the Prototype - https://github.com/unification-com/haiku-node-prototype

- Increasing the value of all actors' data in existing and future marketplaces by building more valuable data packages and optimised offerings.
- Integrating efficient crypto payment gateways to reduce payment fees and improve monetisation potential through new channels.
- Ensuring compliance and standardisation with GDPR, DTP and other standardised models as they are adopted by the industry.
- Acquire new users and increase user engagement through participating in Unified Verifiable Credentials ID system (UVCID).

There are additional future Unification use cases, currently in R&D:

- Optimise the value of existing product and service inventory through Autonomous Machine Learnable Smart Contracts (AMLSC).
- Optimise autonomous smart contracts considering user intent through machine learning.

The combination of HAIKU, Capsule and BABEL enables the opportunity to create a full-cycle revolution by allowing Data Providers to also act as Data Consumers, either within the same cycle or following cycles, allowing all actors - including the organisation, enterprise, and individual - to benefit in different ways. Specific cases covering of all these use cases are discussed at the end of the document.

III. Architectural and Software Components Overview

There are several software components involved within the Unification ecosystem. This section gives a brief overview of each of the main components, and how they pertain to the actors within the system. Each component is explained in greater detail within their individual subsequent sections.

A. *Blockchain*

The blockchain is the central backbone providing a secure and transparent source of truth. The list of permitted Data Consumers and Data Providers are stored on the blockchain, and End Users publish whether they permit or deny the sharing of their data. The blockchain also facilitates remuneration for each actor in the ecosystem with a token called UND.

B. *Smart Contracts*

The particular blockchain needs of Unification are implemented via the smart contracts. These are implemented to facilitate data transfers, UND token transfers and store user permissions, in a secure, decentralised manner. All actors interact with the smart contracts, but only Unification Foundation, the Data Consumers and the Data Providers are required to deploy smart contracts. End Users will not be required to deploy a smart contract.

C. *HAIKU Server Node Software*

The HAIKU Server Node is the software that handles all off-chain state channel processing that is related to a data transfer occurring on behalf of a Data Provider. It interacts with the various smart contracts on the blockchain and the provider's data source(s), in order to process data export, encryption and delivery. All Data Providers within the Unification ecosystem are required to install the HAIKU Server Node software.

D. *HAIKU Client Software*

The HAIKU Client software is the Data Consumer's counterpart to the HAIKU Server Node, and handles the request side of a data transfer for Data Consumers. It also interacts with the smart contracts held on the blockchain. All Data Consumers are required to install the HAIKU Client software.

E. *UVCID DID Updater Node*

The DID (Decentralised Identity) Updater Node is the software within the UVCID infrastructure that handles a DID Owner's requests to update their DID Documents. HAIKU Nodes will have an integrated DID Updater.

F. *UVCID DID Resolver*

The DID Resolver is software used within the UVCID infrastructure, and is responsible for translating DIDs into DID Documents.

G. *BABEL App*

The BABEL App is the front-end mobile appbrowser extension app, which all actors can use to interact with the Unification ecosystem. It acts as a wallet, secure key-store, and front-end to the UApp Store. It can also be used by Data Consumers to initiate data transfers, and by End Users to modify permissions.

H. *CAPSULE (SDKs)*

CAPSULE is a suite of tools and SDKs distributed with the HAIKU software to aid a UApp developer and Data Consumer to configure and interact with their respective HAIKU software. Data Providers and Data Consumers will utilise CAPSULE along with the HAIKU software.

IV. Blockchain

Within the Unification ecosystem, the blockchain is predominantly used to store the important states of the system that need to be - and remain - publicly verifiable. That is, the smart contracts on the blockchain are being implemented precisely for their intended use: as public, transparent contracts between End Users, Data Providers and Data Consumers.

The software entities in the ecosystem - the HAIKU Server Nodes, the HAIKU Clients, and the BABEL App - communicate with and cross-reference each other, via the blockchain, to ensure that each party is acting exactly as it should at any stage of a data transfer process. Since data held on the blockchain is publicly available, it also means that anyone can query it and verify that UApps (both Data Providers and Data Consumers), End Users, and Unification itself are all acting as they have promised to within the ecosystem.

The information held on the blockchain includes:

- User permissions[2]: whether or not a user has granted access to a UApp for their data held in another UApp.
- Metadata Schema: A schema describing the data a Data Provider is making available within the ecosystem.
- UND Rewards: For example, how many UND a Data Provider is asking for the data.
- UApp validity[3]: MOTHER stores the current validity status of a UApp within the ecosystem.
- Data Provider metadata: Such as the HAIKU Server Node RPC host and port, ensuring the entire network is aware of where HAIKU Server Nodes can be contacted.
- Hashes and Merkle Roots required for the construction and validation of DID Documents

Within the Unification ecosystem, the blockchain is not used to store any actual user data. Utilising the blockchain to store data is impractical for several reasons:

1) Data Size: Potentially gigabytes of data may be made available by a Data Provider UApp. Blockchains are arguably not (at the present time) conducive to storing large quantities of data.
2) Immutability: Once information is on the blockchain, it remains on the blockchain. Due to the immutable nature of blockchains, anything written to it will remain there for as long as the blockchain exists. GDPR states that users have the right to have their data erased[4].
3) Public Access: Anyone with Internet access can query a blockchain API endpoint and obtain anything stored on the blockchain as far back as its genesis block.

Even if it were feasible to store gigabytes of data on a blockchain, fully encrypted using the latest encryption methods, the blockchain still wouldn't be the most practical place to store the data. For example, if the encryption algorithm used became compromised in the future, then due to the immutable nature of data held on the blockchain, any data encrypted with that algorithm and stored on the blockchain is also compromised. It would also continue to be publicly available to anyone (since it would not be possible to update the data with a new encryption algorithm). Further, the public nature would make it possible for a nefarious party to download the encrypted data, and attempt to attack and decrypt it at their leisure.

V. Smart Contracts

Three smart contracts are required within the Unification ecosystem. The MOTHER and Token contracts are deployed and maintained by Unification Foundation. The third type of contract is deployed by each individual UApp when they wish to act as a Data Provider and/or Data Consumer within the Unification ecosystem.

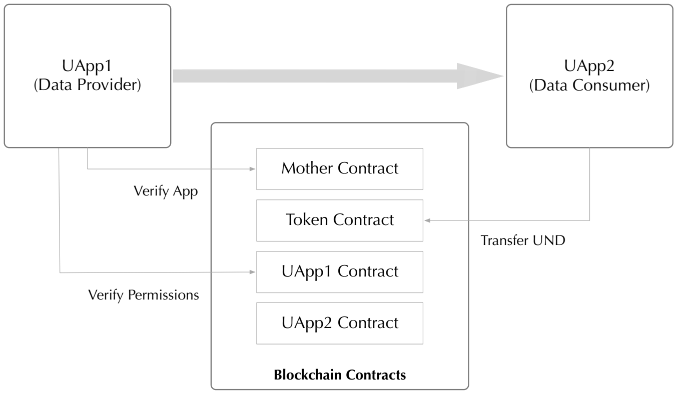

Fig. 1. Smart contract overview

The smart contracts are the crux of the ecosystem, with the network cross referencing and referring to them at various stages of the data transfer process.

A. MOTHER Smart Contract

The MOTHER smart contract[5] is deployed and maintained by Unification Foundation. It is deployed once, and exists as a single instance (that is, only one MOTHER smart contract exists). MOTHER acts as the central information and validation repository for all UApps who wish to be a part of the Unification ecosystem. It serves as the central reference point for data-request validation, UApp discovery, and the basic metadata required for "inter-UApp" communication.

Every HAIKU Server Node, HAIKU Client, and BABEL front end refers to MOTHER for verification of the current state of a UApp. This includes the validity of the app, the validity of the data sources, and the location of the RPC endpoint. Only the MOTHER smart contract owner's account (owned by Unification Foundation) is able to modify any data held within.

Both Data Providers and Data Consumers are only able to act (i.e. transfer data) within the ecosystem once they have been approved and their basic information added to MOTHER.

MOTHER will also be responsible for writing DID Update request tokens to the blockchain, in addition to relaying requests for Verifiable Credentials to the users. It will also act as the central validation repository for valid DID Updater nodes within the network.

1) MOTHER Approval Process: Similar to Google's Play Store, Data Providers and Data Consumers intending to act within the Unification ecosystem are required to undertake a simple approval process. The approval process is important, since Unification Foundation wishes to accommodate only reputable Data Providers and Data Consumers within its ecosystem, and therefore retain the integrity of End Users' data.

The approval process is straightforward, in that once a UApp developer has tested their smart contract and respective

[2] https://github.com/unification-com/smart-contracts/tree/master/unification_uapp

[3] https://github.com/unification-com/smart-contracts/tree/master/ unification_mother

[4] http://www.privacy-regulation.eu/en/article-17-right-to-erasure-'right-to-be-forgotten'-GDPR.htm

[5] https://github.com/unification-com/smart-contracts/tree/master/ unification_mother

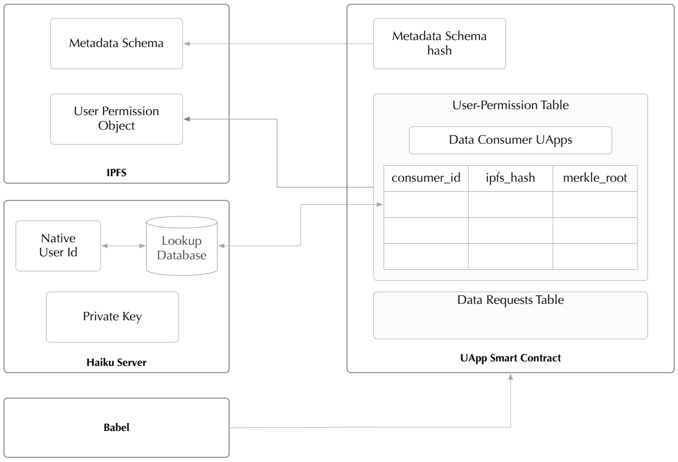

Fig. 2. UApp smart contract

HAIKU Software, the relevant information (for example, smart contract address, RPC Server details and UApp name) is submitted to Unification Foundation for verification by a machine learning optimised process.

The approval decision is based on a number of factors, including for example, the integrity of the UApp's smart contract code - the hash of which is analysed to ensure it matches the expected code hash. The integrity of a UApp's smart contract code is essential, since it is responsible for determining which End User's data is included in a data transfer, based on the permissions held within. The validation process ensures that the code hasn't been tampered with prior to deployment.

Approved UApps are added to MOTHER, and as a result, the UApp Store is updated to include the new UApp.

For any UApp not approved, the developer is notified with a list of recommendations, and given the opportunity to resubmit their UApp for approval.

B. UApp Smart Contract

Each UApp within the Unification ecosystem, whether they are a Data Provider, Data Consumer or both, maintains a smart contract that holds in it:

1) A list of data consuming UApps
2) A permission table of End Users and Data Consumers
3) Metadata describing the data published by the UApp

The End Users of the UApp choose which Data Consumers are allowed access to their data. The metadata provides the Data Consumer the structure of the data feed, and type of data that they can expect.

Only the hashes of the metadata are stored in the UApp smart contract, along with user permissions. The actual metadata is stored in an highly available external data store[6] [7]. Blockchain storage space is an expensive commodity, and storing only the hashes in the smart contract saves the cost to maintain the contract whilst maintaining the integrity of the data.

The UApp smart contract will also act as a relay, validating and forwarding DID Owners' DID Document Update requests to MOTHER.

1) Equable: The UApp smart contract[8] is written such that the UApp developer should not need to write, or even modify any smart contract specific code. Any modifications required will only be to data held within, and will be executed via transactions/action calls (for example, adding or editing Metadata Schemas, modifying access permissions, and initialising or updating data requests).

2) User Permissions: Each UApp smart contract contains a user permission structure. This structure enables a mapping between data consuming UApps and the End Users who have granted them permission to access their data.

A suitable unique identifier is generated in the underlying blockchain, for both the user that is granting or revoking access and the UApp that is requesting the data. This ID is used across the Unification network. A common identifier for End Users and accounts is also required to facilitate UND token transfers.

Off-chain Permissions Storage Storing potentially millions of user permissions within a smart contract is expensive, with regards to transaction costs, RAM costs, or other associated blockchain resources. Using a combination of IPFS and Merkle trees, Unification will store the minimum required data within the smart contract, while the actual user permission data will be stored off-chain, but still be publicly available, verifiable, traceable and only modifiable by the End User. This allows for arbitrarily sized fine-grained permission structures, for large numbers of users to be stored in a relatively small space within the UApp smart contract.

JSON is used to store the fine-grained permission structure, with each Data Provider ¡-¿ Data Consumer relationship being assigned its own individual JSON object containing the list of End User permissions applicable to that relationship. Each End User's permission object within the list stores their blockchain account ID, a nonce, a set of permissions defining the data points for which the user has granted access to the Data Consumer (for example, heart_rate and time), along with a digital signature (derived from the permissions and nonce) and the public key counterpart of the key used to sign the permission object change request.

The JSON is saved by the Haiku Server software to the Data Provider's IPFS node, with the address hash being saved to their UApp smart contract along with the Data Consumer's blockchain account ID. Additionally, the user's stringified permission object becomes a leaf within the Merkle tree, the root hash of which is saved along with the IPFS address hash in the UApp smart contract.

Granting or Revoking access The HAIKU Server Node is required to check whether or not an End User (who has a Unification account ID, and has linked it to the Data Provider) has explicitly granted permission to a requesting app to access their data. If the relationship between the Data Consumer and

[6] To bolster availability, metadata schemas are cached locally, and the hash of the local cache is verified against that on the blockchain. Any updates are processed and cached locally as required.

[7] IPFS is evaluated in the prototype; any fault tolerant CDN will suffice.

[8] https://github.com/unification-com/smart-contracts/tree/master/unification_uappl

End User's Unification account ID does not exist within the Data Provider's End User permission list, then the End User has effectively not given permission, and their data will not be made available to the Data Consumer.

Unification's End Users should not be expected to pay resource costs for granting or revoking permissions to Data Consumers who are requesting data. The proposed method to eliminate costs for the End Users is by a process of signed change requests and signature validation . The End User's BABEL application uses the End User's key to sign a request to grant or revoke access. The request is sent to the Data Provider's HAIKU Server Node's RPC modify_permission endpoint, which queues the request for batch processing. Batch processing is scheduled (for example, with cron). The Data Provider's Haiku server processes each batch, verifying the requests using the user's public key, and only if the signature is valid is the permission modified. The new permissions definition, nonce, signature is updated within the JSON object. The updated permissions are saved to IPFS, the new Merkle tree generated, and the smart contract updated with the IPFS hash and Merkle root hash to reflect the permission changes.

3) Metadata Schemas: The Metadata Schema is used to describe the data that the Provider is making available to the Unification ecosystem. Pseudonyms are implemented to hide the "real" data definitions (for example, database field names), which are reversed by the HAIKU Server Node software via a machine learning algorithm. This algorithm is distributed with the Server SDK as a dynamic tool for developers to generate Metadata Schemas and map these schemas to the underlying sources. This not only allows data to be described in a standardised format, but also hides any potentially sensitive information about the underlying data source(s).

Each data source that a Data Provider wishes to include in the ecosystem must be described, so that End Users, Data Consumers and other Data Providers can view what is available.

In order to reduce potentially high blockchain resource costs for storing larger Metadata Schemas, the schema itself is not stored within the smart contract, but instead a hash of the schema is stored. This hash acts as both a pointer to the actual schema held externally and as a mechanism for validating the externally stored schema.

No private/sensitive data is stored within the smart contract. Only the purposefully publicly available, non-sensitive metadata is stored within the smart contract.

4) UND Rewards: The Data Provider defines how many UND they are asking for a data transfer. Two tiers are definable, for standard scheduled data transfers, and for adhoc data transfers. In the initial phases of the network launch, Unification Foundation will also work with larger strategic Data Consumers to link larger requests & data exchanges outside the immediate system.

C. UND Token Smart Contract

The UND Token smart contract deployed and maintained by Unification Foundation. It is a standard token smart contract with the additional functionality of holding UND during a data request process, and transferring the UND between UApps and End Users once a data request has been successfully completed. The Unification Foundation defines the UND distribution model to calculate how many UND each actor will recieve for a data transfer. Various actors who bring different strategic value to the network can request custom distribution models.

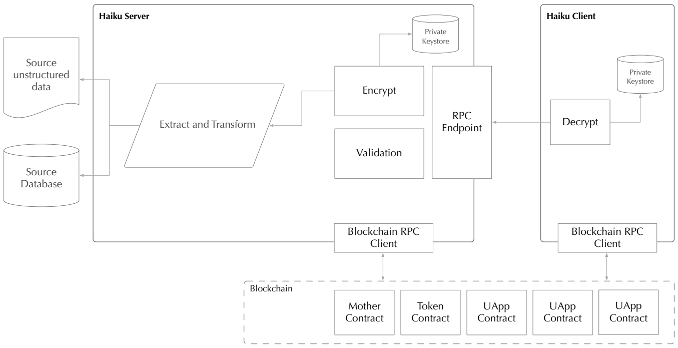

Fig. 3. HAIKU Client and Server Components

VI. HAIKU Server Node

The HAIKU Server Node[9] is a piece of software that all UApp developers who wish to act as Data Providers are required to install. It acts as a middleware component in the Unification ecosystem, facilitating data transfers, by interacting with the underlying data source(s), validating UApps who are requesting data via smart contracts on the blockchain, encrypting data, implementing and processing state channels, and serving data requests via a built-in RPC server.

A. Blockchain Connection

Whilst the HAIKU Server Node is not in itself a "blockchain node", it does have a built-in Blockchain RPC client to interface with the blockchain, the smart contracts, and their respective actions. The server's blockchain client is configurable to connect to the Mainnet via the existing API endpoints, or should the UApp developer desire, connect to a locally running blockchain Full Node. A Unification Testnet will also be made available in due course. Additionally, UApp developers are able to configure and connect to a local, private node for development and testing. These configurations are discussed as follows:

1) Mainnet: Existing RPC Endpoints: Most UApp developers will configure the HAIKU Server Node to connect to one of the existing Mainnet API Endpoints. The configuration tool which will be supplied with the Server SDK, will attempt to automatically detect the nearest API Endpoint based on geolocation.

[9] The HAIKU Server Node is not itself a "blockchain node". It is an independent piece of software that communicates with the blockchain to execute off-chain processes.

2) Mainnet: Local Full Node: The option to run and connect to a local blockchain full node is also supported for UApp developers who wish to do so. This is functionally identical to connecting to existing RPC endpoints, with the exception that should the local full node fail, the HAIKU Server Node will automatically fallback to one of the existing Mainnet endpoints.

3) Unification Testnet: For testing at zero cost, Unification Foundation will deploy a small, closed Unification Testnet. The testnet will be bootstrapped by Unification Foundation for the sole purpose of allowing UApp developers to test UApps prior to deployment on the mainnet.

4) Local Private Chain: UApp developers are also able to develop and test using a local, private blockchain[10]. The current HAIKU Node Prototype, for example, currently runs on a local self-contained, private single node network.

B. RPC Server

The HAIKU Server Node's built in RPC Server is the entry point for actual data transfers between UApps.

By default, the RPC server listens from all addresses to HTTPS requests on port 8050. It serves the following endpoints, accepting POST data:

- /scheduled
- /adhoc
- /verified
- /modify_permission

The RPC Server is launched with either the password to the keystore passed as a command line argument, or as a environment variable. The keystore holds the UApp's RSA private key, which is used for signing and decrypting messages, in addition to encrypting the randomly generated symmetrical keys used to encrypt actual data during the data transfer process. The RPC server is configurable via a JSON configuration file, which is automatically generated by the configuration tool supplied with the Server SDK.

C. Data Extraction/ETL

The HAIKU Server Node is responsible for interpreting data requests, querying data sources, and exporting data into a standardised format. In order to address the complications of the wide range of data storage implementations our adopters may potentially be utilising, we provide a data-export solution that flattens out the data into a block processable by machine learning algorithms.

There are several methods available to flatten common database architectures, ranging from a query-based solution where an indexed query is run to produce a data block, to built-in tools provided by enterprise packages, which export all data or query based subsets of the data.

The motivation for producing a flattened data block is such that the machine learning algorithm (see next section) can process the block and produce a data package in a standardised format. This is accomplished using input parameters supplied by the data producer, indicating which fields hold importance. It also allows the Unification ecosystem to ultimately be "data-source agnostic". This enables the HAIKU Server Node's ETL component to support a number of structured and unstructured data sources out of the box, with more being implemented as the ecosystem evolves.

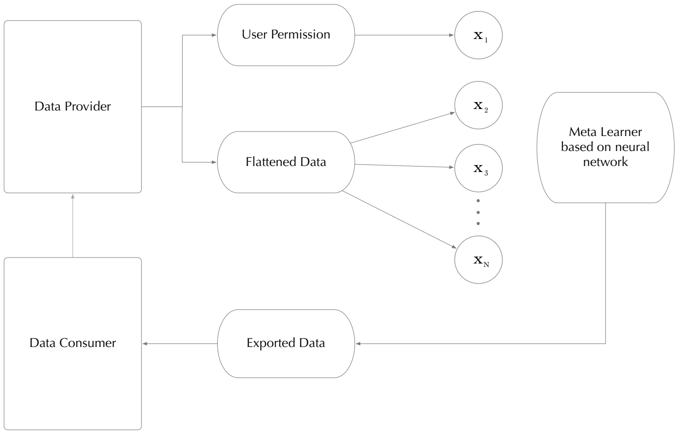

Fig. 4. Overall architecture of ETL process - Phase 1

D. Custom ML Algorithm Mechanisms

A generic algorithm for Data Providers is designed and developed to generate the corresponding Metadata Schemas and map these schemas to the underlying sources without manually hardcoding the linkage. Flexibility is taken into consideration when designing the model. UApp developers, without any machine learning knowledge, can fine-tune the corresponding parameters to systematically fit their data structures. The ultimate standardisation can be achieved in such a way that there would theoretically be no limitations to data provided and consumed by the participants, since each developer can modify the high-level design of the algorithm by finding optimal hyper-parameters of the algorithm to meet their own data requirements. The algorithm is implemented using a neural network, since it has been proven[11] that neural networks have the most capable machine learning architecture when dealing with large amount of various data types. Along with the generic neural network architecture, base weights and bias parameters will also be distributed with the Server SDK.

Mapping between the user's Unification ID and native user ID in each provider's platform is also another output of the algorithm. The efficiency of using a machine learning algorithm resides in the capability of handling large amount of datasets without any prior hardcoding. Mapping between the Metadata Schema and underlying data source(s) has a one-to-one correspondence, meaning multiple database fields only correspond to one user. The machine learning algorithm is initially trained on existing pre-defined schemas and their corresponding data sources (for example using column names as input values, x and user's data as output values, y) to learn on the mapping of schemas to real data. That facilitates

[10] Similar to the current prototype code - a full, sample/demo network will be available for developers to download and implement.

[11] Deep learning applications and challenges in big data analytics https://doi.org/10.1186/s40537-014-0007-7

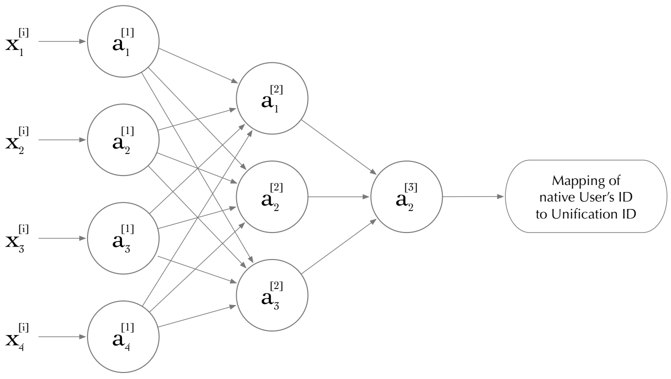

Fig. 5. Phase 2

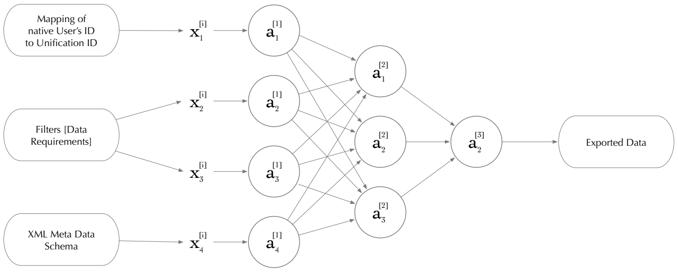

Fig. 6. Phase 3 - Comprehensive Network: Representation commonly used for Neural Networks, where x1, x2, xn in the first input diagram represent fields from source data. For better aesthetic, the parameters such as weights and biases have been omitted.

description generation and, most importantly, linking of user's native ID and users' Unification ID by the algorithm. This occurs in addition to mapping between the data source and user's ID. Furthermore, since every UApp uses the same base and fundamental algorithm, it can be ensured that all data gathered on a specific user from diverse UApps are linked and unified accordingly.

Finally, the machine learning algorithm will be implemented to filter data requests by End User permissions, and Data Consumer queries (where applicable). Each instance of the ML algorithms is described in further detail in the respective following sections.

1) Metadata Schema Templating: The Metadata Schemas (described in the section "UApp smart contract") held within the UApp smart contract are utilised by the ETL component during the data-export process. By passing the schema to the machine learning algorithm, the ETL component is able to map the publicly visible data descriptions to the underlying data.

The Server SDK will contain a tool with a generic ML algorithm integrated which allows UApp developers to automatically generate their Metadata Schemas, and modify the corresponding parameters inside the ML algorithm according to the request requirements.

2) Standardised Data Output: The goal of data standardisation is to enable access to data from a variety of supported data storage formats and to transform the data into a standardised export set, which can be used across the Unification network without requiring further transformation.

Data standardisation occurs at several levels. The highest level is the output format standardisation - that is, the output that Data Providers will produce (for example, CSV). Since all providers will be outputting the same data format, all actors within the network will be able to understand the data they ingest, regardless of the underlying source.

Second is standardisation of data values and data types. For example, one provider may store date and time in ”m-d-y_h:m:s", and a second provider in ”dd/mm/yyyy hh:mm:ss". This is standardised into, for example, a UTC based Unix Epoch integer.

Third, Unification will propose a set of "metadata guidelines", which Data Providers may use in order to aid the definition of a standardised metadata schema. For example, "Heart Rate" field should be defined as heart_rate within the metadata schema.

Finally, User IDs are standardised, with each user being assigned a common blockchain anchored ID for use across the entire Unification network, allowing data from disparate sources to be unified.

3) Machine Learning Algorithm: Any metadata stored within the UApp's smart contract (for example DB field/table names specified in the Metadata Schemas) is stored in the form of pseudonyms. The HAIKU Server Node uses a machine learning algorithm which is fine-tuned (by optimising the "hyperparameters") by the UApp developer to fully meet the data requirements. This algorithm is only accessible by the HAIKU Server Node software, and is utilised to map these pseudonyms to their real values. The generic version of the algorithm is initially distributed with the Server SDK and automatically adapts to the modifications made by the UApp developer when the software and data sources are configured.

Further, the ML algorithm is also used to dynamically map the publicly available blockchain account names used as a Unification User's ID across the network, to the UApp's native User IDs. This not only allows unifying End User data from different sources, but also prevents exposure of the sensitive "real" user ID implemented by the UApp providing the data.

4) ML Algorithm Functionality: The ML algorithm is utilised for multiple purposes and below are descriptions of various stages of training and testing processes of the algorithm:

Dynamic Metadata Schema generation and Data Sources Mapping

The ML algorithm is initially trained on existing pre-defined Metadata Schemas and their corresponding underlying data sources on the first layer of its neural network, in order to learn for future data. An example of this process is using a Metadata Schema as input and the underlying data sources as the output, or target values. When the algorithm is trained and has learned the mapping between the schema and data sources, it will be capable of predicting new data sources based on the fed schemas using previously defined criteria (column names, random number associated to the data sources, etc.). This is used for future mapping when the relationship between the schema and the underlying data is unknown.

Unification ID and Data Provider's native user ID mapping

The ML algorithm is trained, and implemented to identify the link between a Data Provider's native user IDs and the user's Unification ID on the Data Provider's end.

Filtering data based on End User permission and Data Consumer's data request requirements

After the Metadata schema to the underlying data sources mapping is processed by the algorithm, the UApp developers can adjust the hyper-parameters according to the dataset being provided. Any filters based on user's permission or Data Consumer query parameters will be processed by the algorithm before the final data export. The final output of the algorithm can be delivered in a standard format consistent across Unification or transformed to formats required by the Data Consumer.

Merging data from multiple Data Providers

If a Data Consumer requests data from two or more Data Providers in a single request, the ML algorithm will be trained to merge the datasets into a single dataset, in the Haiku Client despite none of the Data Providers having access to each other's databases. This utilises the "pattern recognition" capability of ML algorithm. Each dataset provided has hidden patterns nested (User IDs have been normalised to Unification IDs, Fitbit and Garmin both use Facebook login, timestamp formats have been standardised during the export process) that the ML algorithm is trained on to learn the links between specified datasets can therefore map the data (standardise multiple datasets into one) to an individual user based on common patterns nested in various datasets.

This is achievable due to the data export being standardised into a common format, in addition to data standardisation such as native app user IDs being normalised to Unification IDs. As such, it should be possible to identify the linkage between Alice's data from one Data Provider with Alice's data from any other Data Provider that received a request, and subsequently, consolidate the datasets into one, standardised, and uniform data packet.

5) ML Algorithm Training Process: Initially, the "generic" algorithm is trained based on existing sample data. This reduces the future prediction into a classification problem. During the training process multiple sample datasets are used. This includes information about users, such as their native IDs along with their assigned Unification IDs. Another dataset may contain data (health data, images, Facebook, etc.) that is provided by the Data Provider and contains Unifiction IDs. The Metadata Schema is another dataset which will be used in the training process. During training, an independent test method is used for splitting training and testing data into separate subsets to avoid any overfitting issues.

With the provided Metadata Schema and the underlying dataset, the algorithm uses a deep neural network on the first layer, to learn on the mapping between the two. The neural network consists of a multi-layer perceptrons that split the input into small subsets and perform function approximation. For example, a two-layered neural network is capable of approximating any arithmetic operations and mathematical functions.

The next layer of the neural network consists of feeding the actual data to generate and predict the mapping between the Unification ID and the user's real ID.

For example, the End User "Alice", has the Unification account ID of "37730368". This ID is used across the Unification network to identify Alice, and as such is the ID used for Alice whenever a data transfer or a UND transaction occurs involving her.

Alice has a Fitbit account, and a Garmin account. Alice's Fitbit ID (used internally by Fitbit to identify Alice within Fitbit's database's "users" table) is "123456", and the ID Garmin uses for Alice is "abc456". Both Fitbit and Garmin's HAIKU Server Nodes have a fine-tuned machine learning algorithm which maps these "internal" user IDs to Alice's Unification ID "37730368", e.g.:

```
Fitbit: 37730368 -> 123456
Garmin: 37730368 -> abc456
```

Any request to either Fitbit or Garmin for Alice's data is initiated using Alice's Unification ID "37730368". The HAIKU Server Node software can then use its fine-tuned machine learning algorithm to find the actual data for "37730368". This method ensures that Alice's underlying native user IDs for their respective UApps (123456 and abc456) are never exposed to external parties.

Similarly, all data exports transform any native user ID into Alice's Unification ID using the same machine learning algorithm, ensuring that Alice's data from diverse UApps can be linked and unified.

Example Process Stanford University is undertaking some research, and would like access to some data from Fitbit (heart rate, timestamps) and Garmin (GPS data, timestamps). Alice (Unification ID 37730368) agrees to allow Stanford University to access her data held by both Fitbit and Garmin. Through her BABEL App, Alice has signed permission change requests to both Fitbit and Garmin, whose HAIKU Server Nodes add the permissions modifications to their respective UApp smart contracts. The permission in both smart contracts are recorded under the Unification ID "37730368".

Stanford University sends the data request to both Fitbit and Garmin. Their HAIKU Server Nodes receive and verify the request, then access the permission data held in their respective UApp smart contracts. Both Fitbit and Garmin's HAIKU servers can see that "37730368" has given permission, and their fine-tuned ML algorithm can automatically map "37730368" to their respective internal User IDs (123456 and abc456).

Both Fitbit and Garmin's HAIKU Server's process the data and generate the export for Stanford University, standardising the date format for the timestamps, and Alice's internal User ID to "37730368". Stanford University can then download the data, and unify both datasets based on Alice's Unification ID "37730368".

E. Data Generation

The HAIKU Server component provides data in two data transfer methods: Scheduled, and Ad-hoc.

1) Scheduled: Data is precalculated and made available on a scheduled basis (for example daily, weekly or monthly). The schedule is configured by the UApp developer and written to the UApp smart contract. Everything specified within the Metadata Schema is applied to all user data, and cached in a local secure storage on the HAIKU Server Node. Although cached data is exported for all users, only the data an End User has granted permission for a Data Consumer to access is provided during a data transfer. The cached data is cleared after a specific amount of time to preserve space. If the scheduled feed is insufficient for the data consumer, an ad-hoc request can be made.

Machine Learning Algorithm When a Data Consumer sends a data request to the Data Provider, and once the request and Data Consumer have been validated, the ETL component processes the data to be exported. The ETL component acquires permissions that have been granted by End Users to the Data Consumer for the subsequent processing and filtering. It forwards these permissions, along with the flattened data, to the machine learning algorithm. The algorithm processes the data according to the provided permissions (filtering) and generates the final data export for the Data Consumer.

2) Ad-hoc Requests: Requesting large, arbitrary subsets of data (for example, “heart rates for people from NY in June 2018”) can be resource intensive. The ad-hoc request endpoint supports this need. The HAIKU Server node queues ad-hoc requests from Data Consumers, and processes the data transfer based on a FIFO queue. As with scheduled data, only End Users who have granted access to a Data Consumer are included in the export. Once data has been downloaded and verified by the Data Consumer, it is deleted from the provider’s local secure storage.

The request, export and delivery process is similar to the scheduled data process, with the additional query parameters being forwarded to the machine learning algorithm along with the permissions and flattened data, in order to filter the data as per the Data Consumers requirements.

F. Security and Encryption

The security of data is paramount, and at no point during the data transfer process is the data stored on disk in plaintext by the Haiku software. Data is encrypted as it is exported to the intermediary storage cache. During the data-request process - both scheduled, and ad-hoc - the data is decrypted on the fly and fed into the machine learning algorithms in order to filter the data according to user permissions and any query parameters presented by the Data Consumer.

Support for key rotation and key management on a network separate from the Haiku server (for example a fault-tolerant key management cluster) will be implemented.

The filtered results are then symmetrically re-encrypted using a new randomly generated key unique to each individual data request. This key is asymmetrically encrypted using the Data Consumer’s public key. The encrypted symmetric key is distributed along with the data. When the Data Consumer receives the data package, they are able to decrypt the symmetric key using their own private key, which is securely stored in a password encrypted keystore alongside their Haiku Client software, allowing the Data Consumer to decrypt and ingest the data.

1) Key Storage and Blockchain Account Permissions: A blockchain account is required to transfer or push any valid transaction to the blockchain. Every Data Provider and Data Consumer has an account on the blockchain.

Each Data Provider and Data Consumer blockchain account has a set of custom permissions, linked to specific tasks within the ecosystem and smart contracts (for example, transferring UND, initialising data requests, and updating/adding metadata schemas in the UApp smart contract), and each custom permission level is assigned an independent key. Implementing custom permission levels ensures that higher permission levels and authorisations (for example “owner”) can’t be easily compromised.

Key pairs are generated for each specific task and its associated permission level. Private keys are stored securely within a password encrypted wallet file on the HAIKU Server and Client, and their public counterparts are stored within the respective permission within the blockchain account. Only the keys required for HAIKU Server/Client-specific tasks are stored in the wallet file on the HAIKU Server/Client. Higher permission-level keys (for example, “owner” keys) are never stored within this wallet file or on the HAIKU Server/Client, ensuring that if a lower-level key pair is compromised, the account itself is not compromised, and the affected key can easily be replaced without risk to the blockchain account.

Support for alternative wallet storage locations will also be implemented (for example, allowing developers to store the wallet file on a separate network).

G. Data Request Process

A Data Consumer who wishes to access data from one or more Data Providers will be able to initialise a data request either via their BABEL App, or via custom integrations for larger scale demand-side platforms (DSPs) and enterprise partners. Data Consumers are able to browse the UApp Store interface in the Babel App. The UApp Store listings are derived from a combination of the MOTHER smart contract (which contains information on apps valid within the Unification Network), and each Data Provider’s UApp smart contract (which contains a description of the data they are making available, export schedule, and the amount of UND they require for providing the data).

When a data request is initialised through either the UApp Store, or a custom integration solution, the Data Consumer specifies the requirements for the request, including any query filters (for example, Males from NY during July 2017). The data request information is written to the Data Consumer’s UApp smart contract, and the request is forwarded to the Data Provider’s Haiku Server. Data Consumers will also have the option of setting scheduled tasks (for example, using cron) for regularly scheduled data requests.

The Data Provider's Haiku Server validates the request (see section "Data Request Validation Process" below), and if the Data Consumer's UApp is valid, begins processing the transfer. The Data Consumer's JSON request body contains the Consumer's UApp smart contract address, along with the data request ID, which the Provider's Haiku Server uses to obtain the requirements for the data request.

The Data Provider's Haiku Server generates the data export, which is filtered based on the Consumer's query parameters (if supplied), and whether or not the End Users have granted access for the Data Consumer. Since End User permissions are stored via the Data Provider's UApp smart contract, the Provider queries its own UApp smart contract to check whether or not an End User has granted permission to the Consumer. This is done for each individual data request.

1) Proof of Delivery: Once the data export has been generated, the Provider's Haiku Server symmetrically encrypts the data with a unique randomly generated key (i.e. a key is generated for each data request, and is never reused, as described previously in the section "Security and Encryption"). The key required to decrypt the data is asymmetrically encrypted using the Consumer's public key, and packaged along with the data.

The Data Provider's Haiku Server then updates the data request entry in the Data Consumer's UApp smart contract, with a checksum generated from the encrypted data[12], and the Data Consumer then downloads the data package.

Once downloaded, the Data Consumer's Haiku Software computationally validates the data with the provided checksum. Computational validation ensures that the data has not been corrupted or tampered with during the request, transfer or download process. Since the checksum is written to the smart contract, it also allows for transparency, and can be used during the arbitration process should a dispute arise.

Following the computational validation, the Data Consumer may implement their own internal data validation process to ensure that the data they have received is the data they requested. At this stage, if the data has not been delivered as promised, or is not what the Data Consumer expected, it will be possible for the Data Consumer to raise a dispute, in which case UND payment will not be released (or refunded) until the dispute has been resolved.

Once the data has been validated, the UND are released and payment forwarded to the Data Provider and End Users.

2) Data Request UApp Validation Process: Every data request coming into the HAIKU Server Node is validated, starting with the initial signature of the RPC request to ensure that it is a genuine request from the Data Consumer. The UApp validation process is mandatory, and the first thing that occurs when processing a data request.

Once the signature for the request has been validated, both the actual Data Consumer's Unification ID and its associated UApp smart contract are validated against the MOTHER smart contract. This validation process involves checking that the UApp is valid according to MOTHER, as well as verifying that the Data Consumer's UApp smart contract is valid. This validation process occurs by checking the hash of the deployed smart contract code against the hash stored in MOTHER at the time of UApp validation/deployment, to ensure the contract code hasn't been tampered with and re-deployed.

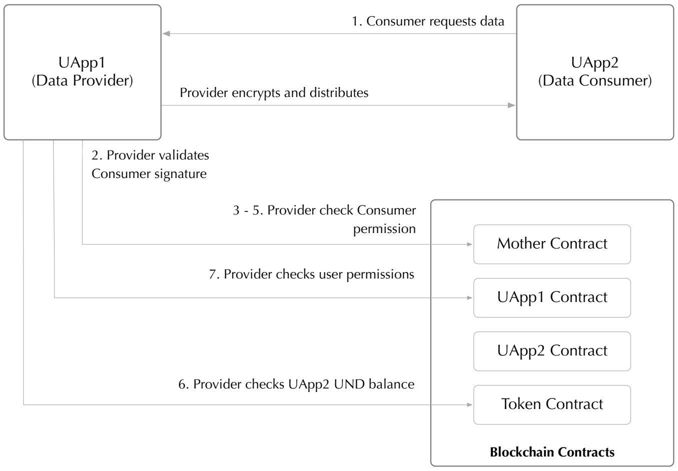

Fig. 7. The data validation process

The validation process is currently executed in several stages:

1) "UApp2" sends a data request to the "UApp1" data_request RPC endpoint.
2) "UApp1" checks the incoming request signature, to ensure that it has indeed come from "UApp2".
3) "UApp1" then "calls MOTHER" to ensure that "UApp2" is fully valid, and allowed to request data.
4) "UApp1" then queries the Blockchain RPC to obtain the current hash of the "UApp2" smart contract, currently held on the blockchain.
5) "UApp1" compares this code hash with the hash stored in MOTHER to ensure that they match.
6) "UApp1" checks that "UApp2" has forwarded the UND payment to the Token smart contract.
7) Finally, if all validation steps are successful, "UApp1" queries its own UApp smart contract permissions structure to see which users have granted access to "UApp2" and serves the data accordingly.

If any of the steps #2 to #6 fail, the request is rejected and the Data Consumer is informed that no data is served.

H. Unifying Non-Unification Users

Not all End Users within a Data Provider's data source will immediately have an associated Unification account and ID. Hence temporary IDs are assigned to each user in the underlying UApp. For example, if the UApp utilises social login APIs (such as Facebook Login), the ID can be used as a basis for generating the temporary Unification ID. The social login ID is hashed, along with a salt, and stored within the HAIKU Server. The salt is encrypted and packaged along with the data, allowing a Data Consumer to potentially unify "non-unification user" data from disparate sources.

1) Temporary UND Holding: UND Rewards can be accrued by a user even if they do not yet have a Unification ID. Any UND rewards which should be paid to an End User,

[12] using a cryptographic hash algorithm such as SHA-3

and for which the End User's Unification account does not yet exist, are instead held on their behalf. Upon such a time that the user creates a Unification account, and links it to the UApp native account, the accrued rewards are immediately forwarded to the End User's Unification account.

The tokens will be held within the Token smart contract. When a Data Consumer initiates a data request, they also forward the UND, as specified by the Data Provider, to the Token smart contract. Once a data request has been successfully completed, the smart contract action to release the funds is executed, and the Token smart contract will forward the UND according to the distribution model in place.

For example, if the Data Provider has specified that the cost for the data is 100 UND, and the distribution model in place specifies that 60

VII. HAIKU Client

The HAIKU Client is a piece of software that Data Consumers are required to install. It acts as the bridge between the BABEL front end for Data Consumers, the UApp smart contract (which Data Consumers are also be required to deploy), and the HAIKU Server Node(s) to which it makes data requests.

The Client software automatically handles initialising data requests on behalf of the Data Consumer (which are initiated via the BABEL Front end for Data Consumers). It compiles and signs requests to the HAIKU Server Node's data delivery RPC endpoints, forwards UND payment to the Token smart contract, then downloads and verifies the data. Once verified, the client then initiates the UND payment forwarding to the Data Provider and End Users via the Token smart contract.

As with the HAIKU Server, the Client software has a built-in Blockchain RPC Client, which can connect to the blockchain via the same methods outlined in the HAIKU Server section.

A. Data Request Validation

Similar to the HAIKU Server Node data request validation process, the HAIKU Client also validates Data Providers before generating and sending a data request. The process is almost identical, but reversed, with "UApp2" validating "UApp1".

VIII. BABEL Front End

The BABEL front-end will be available in End User, Data Consumer and Data Provider editions, all built around the same core wallet and identity functionality.

A. Wallet

BABEL offers the standard wallet functionality, allowing all actors to view their UND balance and transaction history, as well as to send and receive UND.

1) Keys and Security: When an End User signs into Unification for the first time, a blockchain account is created on their behalf. The private key is stored securely in the BABEL app on the End User's mobile device, in a password encrypted file. In addition, a private key recovery process is provided to the End User. The BABEL App itself is secured with a PIN.

B. User Permission Control

Once an End User has linked their Unification account to a UApp, they are able to view any data requests within the BABEL app. The End User is then able to grant or revoke access to Data Consumers to their data held by a Data Provider.

C. Data Overview

By querying the information held in the smart contracts (Metadata Schemas, data requests and End User's permissions), BABEL offers an overview to End Users about who is accessing what data.

D. UVCID Control

End Users will be able to store and access their Verifiable Credentials via BABEL. In addition, they will be able to view VC requests, and either grant access, generating a Verifiable Presentation for the requesting Verifier, or deny access.

End Users will also be able to maintain their DID Documents.

E. Data Consumers

The search functionality for Data Consumers will provide tools to allow them to locate potential data sources for their needs, ranging from broad metadata searches to detailed data field info.

Data Consumers link their BABEL app to their HAIKU Client software in order to initiate and facilitate data requests and exchanges.

F. BABEL UApp Store

Data Consumers can view a list of valid apps in the BABEL front end. Whether an app is valid or not, is stored in the MOTHER smart contract. Hence, the BABEL frond end queries MOTHER to obtain this information.

Categorisation and other data (UApp title, description etc.) is held off-chain in a database on the Unification servers. BABEL also queries each individual UApp's smart contract to obtain information on what data is being made available.

IX. UVCID

Unification's Unified Verifiable Credentials ID (UVCID) system provides the ability for both Enterprise and End User's to request and declare Verifiable Credentials[13]. A Verifiable Credential is a document signed by an Issuer, than can later be cryptographically verified. Not only does this allow End Users to own and control personal information and documents within their BABEL app (in addition to allowing them to control the flow of their data between UApps), it also provides the ability to re-use previously issued VCs.

The ownership of the Verifiable Credentials that UVCID provides is built upon the principles of Decentralised Identifiers[14].

[13] Verifiable Credentials Data Model: https://w3c.github.io/vc-data-model/

[14] Decentralised Identifiers (DIDs): https://w3c-ccg.github.io/did-spec/

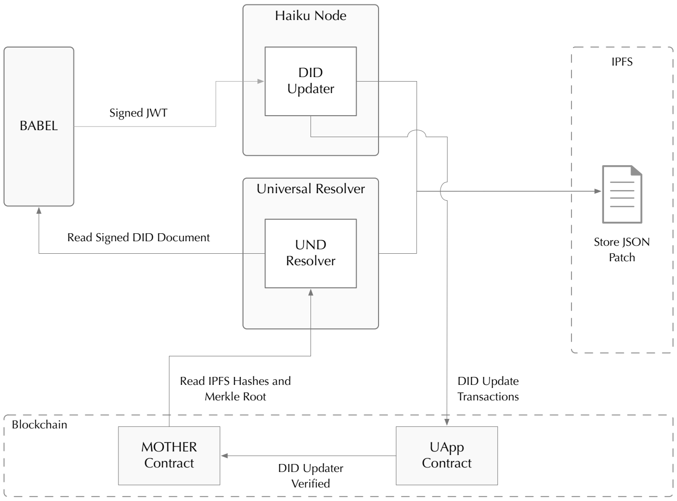

Fig. 8. DID updates and resolution

A. The DID and the DID Document

A Decentralised Identifier (DID) is a pointer, or "key" to a DID-Document, which itself is a JSON object in JSON-LD format that can be proved to be owned by the DID Owner. A DID Owner within Unification can be any actor - an End User, Data Provider or Data Consumer.

When the End User first registers with the Unification Foundation, via BABEL, a DID Document is created on their behalf, for their associated blockchain account. The DID Document is identified by their blockchain account, with the "did:und" prefix. The End User is granted ownership over this DID by associating their blockchain public key with it.

1) DID Document Construction: DID Documents within the Unification ecosystem are based on a blank DID Document template, as defined by the DID Document specification[15]. Any change to the base document is encapsulated by a JSON Patch[16]. For example, the additional of the End User's blockchain public key, will be encapsulated by a JSON Patch that modifies the appropriate "publicKey" and "authorization" sections.

A balanced Merkle tree is constructed from these JSON patches to provide decentralised integrity of the overall DID Document. In addition, each JSON patch contains a reference to the Merkle tree of the previous operation. Each JSON Patch is stored in IPFS, and the location of this patch in IPFS, along with the Merkle root, is appended to the blockchain.

2) Processing DID Updates: Updates to DID Documents will be processed by DID Updater nodes within the Unification Network. This functionality will also be included in the Haiku software. As with Haiku nodes, these nodes will be anchored to a smart contract, and will go through a similar Unification controlled validation process before being accepted into MOTHER.

Users' requests to update the DID Document, will be generated via BABEL in the form of a JSON Web Token (JWT). The JWT will be signed using the private key of the DID Owner. This JWT will be forwarded to one of the DID Updater in the network for processing. The node will decode and validate the token using the counterpart public key held in the DID Document, to ensure the request has come from the owner of the DID Document, before saving to IPFS, generating the Merkle tree, and passing the request to its own smart contract. The smart contract will in turn execute an action call to MOTHER, which will undertake the final checks to ensure the calling node's smart contract is valid, before writing the DID Update request to the blockchain.

3) Spam Prevention and DDoS Mitigation: To prevent rogue elements spamming the network, all DID update requests will be subject to a size cap, and the number of DID update requests per blockchain account will be limited to, for example, one transaction per second.

4) DID Resolution & DID Read: DID Resolution is the process of returning a DID Document from a queired DID. The "did:und" namespace is reserved for accounts held under the Unification Foundation. The reference implementation of DID Document Read is as follows:

The blockchain account described by the DID is read sequentially for all DID modification operations. The result of the query is a list of tuples containing the Merkle root and hash of the location of the JSON Patch in IPFS. The JSON Patches are obtained from IPFS, and are applied to the base JSON document, to produce the resultant DID Document. Finally, a Merkle tree is recomputed from all the JSON fragments, and compared against the latest Merkle root.

B. Verifiable Credentials

A Verifiable Credential (VC) is a set of data, or a digitised document which has been digitally signed and issued by an Issuer. The contents of a VC are typically about an individual End User. A VC may contain a claim such as "Alice is over 21", a digitised document, such as a Driver's License, or both. VCs will be generated and issued within the Unification system, and sent to End Users for them to store via BABEL. End Users will be able to countersign and distribute VCs in the form of a Verifiable Presentation.

Verifiable Credentials are linked to an individual's DID. The DID is an integral part of the VC, and is one of the elements included in the digital signature. The ownership of the VC can be cryptographically proven via the DID.

1) VC Schema: Similar to Metadata Schemas published by Data Providers, Issuers within Unification's UVCID system will be able to publish VC Schemas, which will contain metadata describing the VCs they issue, and the "type", which will be the identifier for a particular VC within the ecosystem. This will enable Verifiers and Data Consumers to clearly understand the contents of a particular VC they are receiving, in addition to generating requests for a particular type of VC.

Unification will publish a set of guidelines outlining best practice for details such as data field names and type names.

2) Verifiable Presentations: When an End User wishes to distribute a Verifiable Credential, they will generate a Verifiable Presentation (VP)[17] using BABEL. A Verifiable

[15] Decentralised Identifiers (DIDs): https://w3c-ccg.github.io/did-spec/

[16] JSON Patch: https://tools.ietf.org/html/rfc6902

[17] Verifiable Presentation: https://w3c.github.io/vc-data-model/#presentations-0

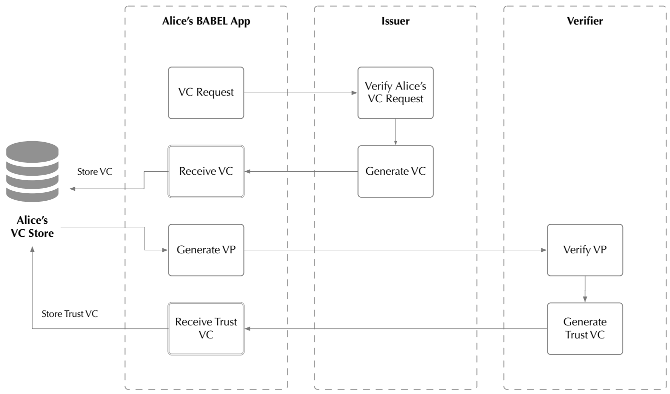

Fig. 9. The VC Web of Trust

Presentation is a JSON Document which is comprised of one or more enclosed Verifiable Credentials, and is counter-signed with the particular End User's blockchain private key (for which the public counterpart is contained within their DID Document).

The Owner's DID is embedded within the VP, and is one of the elements used by the owner when digitally signing the VP, in order to cryptographically prove ownership. As long as the private key used to sign the VP has its public counterpart within the DID Document, ownership can be proven.

3) Requesting Verifiable Credentials: A Verifier will be able to request a Verifiable Credential from an End User via the MOTHER smart contract. The Verifier will generate a JWT, and call an action within the MOTHER smart contract, which will write the request to the blockchain history. The transaction will include the DID for the End User, and the Type of Verifiable Credential.

When an End User accesses their BABEL App, it will scan the blockchain history for any VC requests. If the End User currently holds the requested VC type, the request will be decoded, verified, and displayed for the End User to take appropriate action.

4) Verifications: Any Verifiable Credential or Verifiable Presentation that is accessed or distributed via BABEL, can be cryptographically verified. The DIDs for the owner, issuer and the counter-signers are resolved, and the public keys of these entities are obtained from the DID Document. Each signature is verified using the public keys. A detailed explanation of how the document was verified is also provided to the End User.

5) The Private Store: The facility for End Users to store Verified Credentials is provided via BABEL. The actual store, whether it is Google Drive, S3, IPFS, Archon etc used is agnostic to the Unification Foundation, and is decided by the End User. A SQLite databases exists within BABEL to track the location of each stored document.

6) The VC Web of Trust: Similar to the the validation process required for Data Providers and Data Consumers prior to becoming part of the Unification Network, Issuers and Verifiers will be required to go through a validation process.

When a Verifier receives and verifies the credentials contained within a Verifiable Presentation, they will have the option to countersign each Verifiable Credential to state that they have verified, and trust the signature of the original VC. They will effectively issue a new VC, referencing the original VC, which can be sent back to the Holder. For any future requests for the original VC, the Holder has the option to create a Verifiable Presentation containing not only the original VC, but the additional "trust" VCs generated by each Verifier.

X. Server CAPSULE (SDK)

The HAIKU Server Capsule (SDK) is a suite of tools that will be distributed with the HAIKU Server Node software to aid a UApp developer in installing and configuring the server software and data sources. The machine learning algorithm and associated tools for fine-tuning will also be distributed with the SDK.

A. Support Tools

1) Config Tool: A general configuration tool, which generates and edits the JSON configuration file used by the server. It contains details such as the Blockchain RPC server, UApp's smart contract account name, and so on.

2) Metadata Schema Designer: The Metadata Schema Designer is a tool that will also be distributed with the server software. The tool allows the UApp developer to connect to data sources and to define the data they wish to make available to the ecosystem. The tool implements a machine learning algorithm to automatically generate the Metadata Schema for publishing in the developer's smart contract.

3) ML Algorithm Fine Tuning Tool: The required tools for enabling the developer, without any prior machine learning knowledge or experience, to fine tune the machine learning algorithm implemented in the ETL component. When using deep neural networks, a number of hyper-parameters need to be adjusted according to the size of data, data requirements and filters, etc. Hyper-parameters will be used by the developers in the future to fine-tune the algorithm to meet their data requirements. This way, the generic algorithm is tailored to each Data Consumer's requirements. These hyper-parameters are categorised into two main groups:

1) Hyper-parameters related to the network structure:
   a) Number of hidden layers
   b) Number of hidden units
   c) Dropout agents
   d) Network weight initialisation
   e) Activation function
2) Hyper-parameters related to training:
   a) Learning rate (decaying)
   b) Momentum
   c) Number of epochs
   d) Batch size

The algorithm is initially designed and developed with a default hyper-parameter settings based on the training datasets.Later on, each developer modifies these parameters when dealing with different datasets.

4) Smart contract Deployment and Interaction: Tools to aid a developer in deploying the smart contract, and to simplify the process of interacting with smart contract actions (i.e., writing the Metadata Schema to their smart contract).

XI. Client CAPSULE (SDK)

A Client App Capsule (SDK) will also be made available, which can be integrated into the UApp developer's native mobile/web app. The SDK facilitates communication between the native app, and the HAIKU Server Node, allowing integration of the Unification Single Sign on API, in addition to linking a User's Unification ID with the App's native user ID. The SDK will also provide any additions required to the App's End-user license agreement for future compliance. The SDK will be available for the major platforms and languages (Android, iOS, JavaScript, PHP etc.)

A. Unification Single Sign-on

Similar to many Social ID Login services, Unification will also offer a single sign-on service, allowing UApp users to log into Apps using their Unification account credentials.

XII. Use Cases

Unification's decoupled architecture toolkits bring machine learning and efficient, transparent data transfers together. This innovation allows us to develop autonomous applications that will revolutionise enterprise and improve each of our everyday lives.

The following are a few cases where these can be useful:

Building more efficient data transfers via state channels internally among different business divisions to optimise partnerships and improve user product experience.

In the case of smaller companies, it is common occurrence for various divisions to utilise different external SaaS tools, many of which fit their team's use case best but are not cross-compatible across the business overall.

It's often the case that an enterprise faces this growing pain as divisions get large enough to require custom work among different teams. (For example, a medical clinic that needs to coordinate its internal databases with that of a new insurance system it has recently joined.)

In this case, an internal state channel will allow standardisation between similar data types without the need to rewrite either database.

The same can be applied to various government divisions that hold information on citizens, including tax, civil, criminal and motor vehicle divisions. Utilising Unification's state channels, all departments can standardise and transport their data effectively between offices, while maintaining the transparency and accountability required by citizens. (Note: Although back-end systems that tie this information together do exist, none enable the level of standardisation that meet the transparency and accountability requested and soon to be required from citizens.)

In addition, enabling data connections while removing the centralised point of control for public data will enable efficiency while drastically reducing threats, such as the cyber attack in Atlanta earlier this year[18].

Building more efficient data transfers via state channels externally with partners to optimise partnerships and improve user product experience.

While there are infinite possibilities of data transfer partnerships that can be useful among enterprises, we can review some of the most basic examples here to demonstrate simple feasibility.

Letters of Credit

Credit verification in many developing countries for large scale purchases is still a manual phone, fax or paper-trail process taking up to 10 days for verification of documents, assets, or accreditation of an individual. Unification's data interoperability and UVCID systems are working on a test example to fully automate this process and bring the time from request to verification to minutes.

Our initial test networks is being development for implementation between national auto dealers and banks in the largest developing economy in the Americas.

Insurance Verification

Up to 40% of insurance operational fees are spent on claims verification and false claims corrections in developing countries.

Unification's data interoperability and UVCID systems will allow insurance companies and hospitals to work together to reduce work or costs for hospitals by automating and therefore drastically reducing the process of approval or denial which are now handled manually.

Makes options easier for patients by knowing where or what services are available to them and possibly lower premiums by choosing better options aligned with the insurance or hospital poptions.

Allow hospitals to better market themselves due to what's possible or available for patients and use patient data from other verifiable sources or credentials (for example a Fitness app) to automate.

Airlines, hotels and transportation

Given the complementary nature of hospitality businesses and the amount of redundant inventory that is the norm in that industry, it's possible for these businesses to build internal data transfers whereby remnant inventory is offered across partners to End Users who have the right assumed intent, determined by itinerary, GPS locations, and financial information. In this instance, user affiliations or interests (both declared and assumed) based on the user's profiles can be incorporated to build the best offer for both them and businesses.

For example, a golf course that has a cancelation or open inventory can offer the excess inventory of a particular tee time.

[18] https://www.nytimes.com/2018/03/27/us/cyberattack-atlanta-ransomware.html

The golf course will initially offer that inventory to its internal customer base. However, by building a larger exchange via a state channel with other hotels, airlines, car rentals agencies, etc, they can also extend that offer to other users outside their immediate customer list who have previously registered interest or activity about golf, or who may display a higher spending pattern.

Medical Research

23andMe has recently sold an undisclosed stake to GlaxoSmithKline for $300 million USD. While we're not aware of the terms of the deal, we do know that the 23andMe's user base is approximately 5 million, putting the approximate user value at $60[19].

Details regarding the exclusivity of this partnership have not been announced and it's a wise assumption that there are many other research institutions that could benefit from micro-data transfers with 23andMe. However, given the nature of the information, it's not efficient for smaller organisations to invest the effort of mapping, standardising and transferring this data.

Utilising Unification's HAIKU Server Node, 23andMe can build specific bonds with different organisations within a standardised sharing approach. This would allow new monetisation potentials for Data Providers and increase the amount of data available for both for Data Consumers, benefiting both actors.

Increasing the value of all actors' data in existing and future marketplaces by building more valuable data packages and optimised offerings.

Medical Research

Unification's data standardisation, transportation and data transfer will enable the possibilities of increasing data value for all Data Providers through aggregation into more valuable data packages. New user combinations can be created by Data Consumers, by combining and transporting data through various complementary sources, to create the optimal audience for a given program (research, advertising, etc).

Additionally, by using Unification's HAIKU Node and BABEL, Data Consumers can more effectively request new user information that is not currently available. This is accomplished by issuing a compensation level satisfactory for the Data Consumer as well as the End User.

Example Stanford Hospital is looking for research participants to provide specific details such as Heart Rate, GPS location, and DNA. All of these can be purchased via Unification Data Provider partners in BABEL. Each of these Data Providers currently earns an average of $1CPM for their data, but Stanford is willing to pay $100CPM for individual users.

Unification's system can combine and transport standardised data from these users to allow a higher sell rate for Data Consumer, a lower pay rate for Stanford, and compensation to the End User all at the same time.

For the next phase, Stanford may seek a datapoint not available in Unification's platform, such as users matching the criteria above that also have a specific blood type. It can issue a request for those specific users through BABEL. Specifying the higher amount they are willing to pay for those matching user, users matching that intent have greater incentive to respond.

Coming back to the idea of full-cycle revolution, beyond the initial compensation, Stanford can also provide custom reporting to their users via the HAIKU link, either immediately or years later, as long as the HAIKU is maintained. (This link that would normally be erased via a cookie, which has a maximum life of 90 days.)

Insurance Approval

The insurance industry relies heavily on data to determine risk profiles—and thus rates—for its customers. However, until recently, insurance companies have had to rely on data that was heavily prone to error, including misreporting or misrepresentation by consumers.

These companies have also had significant challenges accessing the data that would most reliably predict whether or not a customer is likely to file a claim. For instance, they have not had ready access to real-time data about a customer's physical activity levels (in the case of health insurance) or braking speeds (in the case of auto insurance). The result is that they have had to make risk assessments by triangulating from other demographic factors, such as age and gender, that have less bearing on the risk of any particular individual.

Using Unification's HAIKU Server Node, insurance companies are able to acquire data from new sources, many of which may have a higher correlation to a consumer's actual risk profile than previously available. As a result, they are able to adjust insurance prices to match the real risk presented by each individual customer, rather than pricing based on demographic blocks.

In this scenario, people who exercise regularly or brake more slowly will have their insurance rates lowered to reflect the actual risk of insuring them. This leads to fairer pricing for all, and helps to insulate insurance companies from the risk of undercharging customers who actually present significant risk of filing claims.

Integrating efficient crypto payment gateways to reduce payment fees and improve monetisation potential through new channels.

Any business that currently deals with large payments has options of processing through bank or wire transfers, merchant payments, or smaller payment processors such as Stripe and PayPal.

There are broad inefficiencies with all options, further amplified when considering internationalisation of the payments payee and recipient. These play out in high transaction costs, increased risk of potential failure or intervention by centralised organisations, fraud, etc.

The addition of crypto payment gateways can increase efficiency and the viability of maintaining a customer base, while most importantly lowering payments for both sides. Unification will not require actors to change their user behavior, but provides additional options to further enhance efficiency.

[19] http://time.com/5349896/23andme-glaxo-smith-kline/

Enabling compliance and standardisation with GDPR, DTP and other standardised models as they are adopted in the industry.

In order to share information within compliance and accuracy standards, organisations will be forced to conform with many new standards. GDPR and DTP are two examples of these standards that have already been implemented or are in process for future implementation.

By utilising Unification's platforms, partners can automatically retain compliance with current and future standards.

UVCID

Acquire new users through participating in Unified Verifiable Credentials ID system (UVCID).

Unification's Unified Verifiable Credentials ID system (UVCID) is built to benefit all participants but is particularly useful to enterprises with user bases between 500k-10M.

Enterprises of this size typically rely on user login systems with no network effect or cross compatibility, or those built by organisations like Facebook and Google who compromise the control, ownership and transparency of the datasets.

Even in the use of the login systems mentioned above, there is very limited to no verifiability or transparent Web of trust to optimise between various actors given the lack of transparency of the central controlling authority.

Utilising Unification's UVCID, enterprises and organisations can offer an alternative to users that provides a similar ease of use without a compromising external central authority.

In addition, utilising UVCID will enables network effects of users across all other UND partners, allowing easier expansion through network effects.

Future Use Cases of Unification Currently in R&D

Optimise the value of existing product and service inventory through Autonomous Machine Learnable smart contracts (AMLSC) and Optimise Autonomous Smart Contracts considering user intent through Machine Learning

While previous examples have covered use cases where Unification's decoupled architecture can help existing business processes, the emergence of AI within our framework can allow us to introduce Autonomous Machine Learnable Smart Contracts (AMLSC). In a very practical sense, Unification can help lead the way by allowing smart contracts find the most efficient scenario matches, based on Autonomous Machine Learning. Examples include:

- A service business setting parameters to allow excess (but expiring) inventory to move for a lower price instead of expiring (for example, a hotel, or a car rental business). This is very similar to current models applied to online ad inventory, etc, but the idea of applying it to real-life scenarios requires all parties to declare user intent (ie. for those that are willing to purchase the service in real time). While this is more difficult on the individual basis, the combination of the sharing economy, AI and smart contracts can bring efficiency to reality for brick and mortar businesses in the same efficiency as the online marketplaces.
- In an autonomous system combining user intent to save time or money with a self-driving car service, Autonomous Machine Learnable Smart Contracts (AMLSC), will optimise the needs of both actors and a match will be made. Users can declare interest to save time or money by switching to a different less busy route, in effect "selling" their spot in the route to another party by having their vehicle pull over to allow someone else to go ahead in traffic. In return, they receive compensation within their declared interest level.

XIII. Glossary

AMLSC Autonomous Machine Learnable Smart Contracts.

BABEL Front End The UND wallet and associated front-end app(s) for End-Users and Data Consumers.

BABEL UApp Store Front-end interface to the UApp Store, rendering a Unification-curated listing of valid Data Provider UApps and the nature of the data they are providing.

BABEL UApp Store: Front-end interface to the UApp Store, rendering a Unification-curated listing of valid Data Provider UApps and the nature of the data they are providing.

CAPSULE: Client and Server SDKs distributed by Unification.

Client SDK: The SDK supplied to facilitate existing app (mobile/web/browser extensions) integration into Unification.

Data Consumer: An entity or organisation who develops a UApp with the help of the HAIKU Server software in order to consume data being made available from Data Providers. A Data Consumer can also be a Data Provider.

Data Provider: An entity/organisation who develops a UApp with the help of the HAIKU Server software in order to provide data to the Unification ecosystem. A Data Provider can also be a Data Consumer.

DID: Decentralised Identifier.

End User: An individual (e.g. Alice) who has been providing data to an application (e.g. Fitbit), and is incentivised within the Unification ecosystem to share their data with Data Consumers.

HAIKU: The concept of a user's "sovereign data store", made possible with the HAIKU Server/Client software, and the smart contract web.

HAIKU Client: The software implemented to execute the client-side off-chain state channels.

HAIKU Server Node: The software implemented to execute the server-side off-chain state channels.

Issuer: An entity who can create and sign a Verifiable Credential.

Holder: An entity who holds and stores a Verifiable Credential. A VC which they hold usually, but not always, contains information about the Holder.

Metadata Schema: Data describing the structure of data offered by a Data Provider.

ML: Machine Learning

MOTHER: Smart contract deployed and maintained by Unification Foundation, holding a curated list of valid UApps within the system, and associated metadata.

NN: Neural network

Requesting App: The Data Consumer's UApp, and respective HAIKU Client software and smart contract.

Server SDK: The Server SDK and tools supplied with the HAIKU Server Node software.

UApp: "Unified App"- any Data Provider or Data Consumer who implements the HAIKU Server Node and smart contracts or HAIKU Client in order to become part of the Unification ecosystem.

UApp Store: The UApp Store content is generated by querying metadata held within the smart contracts used within the system, and is accessible via the BABEL App (the front-end mobile app/browser extension for the system).

UND: "United Network Distribution" - The token central to the Unification economy.

Unification ID: The underlying, backbone blockchain account ID/wallet address upon which the Unification account is built. All End Users, Data Consumers and Data Providers require a Unification ID.

UVCID: Unified Verifiable Credentials ID system

VC: Verifiable Credential. Digitised information which can be digitally signed, and cryptographically verified.

Verifiable Presentation: A collection of one or more VCs, usually created by the holder of the VCs, and sent to a Verifier.

Verifier: An entity that can verify the contents of any given Verifiable Presentation, through cryptographic proof of the VC's signature.