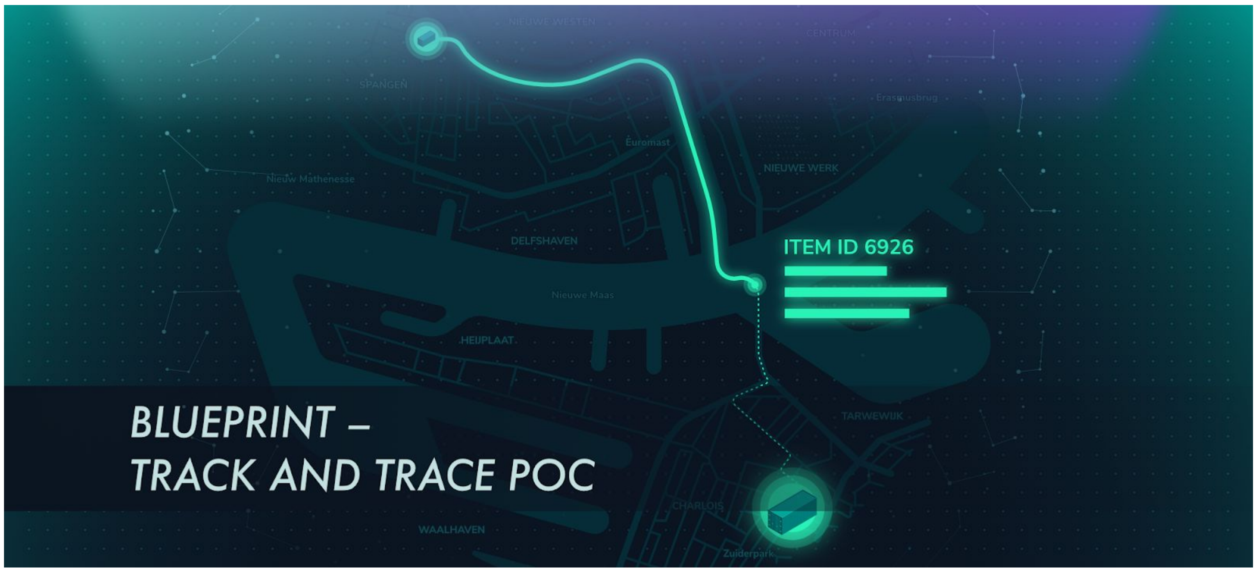

Version 1.0 - 07/01/2019 - Michele Nati & Jens Munch Lund-Nielsen

Introduction

The IOTA Foundation (IF) is a non-profit organization supporting the development and adoption of the IOTA Tangle, a permissionless Distributed Ledger Technology (DLT), particularly suitable for creating trusted information and value sharing across multi-stakeholder ecosystems. IOTA technology is open source. IF's agile approach to solutions creation leverages on identifying industry problems with real stakeholders, building PoCs and collaboratively testing and validating or refining assumptions.

A blueprint is a document explaining how IOTA technology is used to solve real problems and to support well-defined business needs. It should work as a reference for others to replicate and deploy the developed solution into similar systems and to test different business models.

A blueprint is generally easy to read and understand, but nevertheless, it provides enough clarity on how easy it would be to integrate the described solution into other systems. At the same time, it contains links to more technical resources available to developers.

This blueprint addresses a real *track and trace* problem. Track and trace among multi-stakeholder ecosystems faces issues of trust. When sharing information about shared assets confidential information could be leaked about each stakeholder distribution chain and value of market. To solve this issue, the use of a distributed ledger is advisable.

The document is organized as follows:

- We first describe the problem worth solving, underpinning the technical as well as the business challenges associated to it and the benefits deriving from developing a novel solution;
- We then present a technical architecture showing how a PoC leveraging IOTA Tangle and other IOTA technologies can be built and integrated with existing systems. The architecture aims to highlight how easy it is to integrate IOTA with existing legacy systems.
- We finally describe source code showing implementation patterns used and replicable for the integration of IOTA in a similar context. We also provide links to accessible resources useful to simplify deployment of the presented PoC.

Use case

This blueprint describes how a track and trace system for recovery of *returnable assets* has been implemented by the IBCS Group, using the IOTA Tangle and other IOTA technologies. Integration of this system within IBCS Group's business processes is presented in order to provide a guide for other organizations willing to replicate and adapt the produced software to scenarios and business processes similar to the ones presented.

A *returnable asset* is an asset (e.g. a container or in this blueprint a glass rack) used in the distribution and logistic chain for the handling and delivery of other valuable assets (e.g. glasses).

In the case of the glass manufacturing industry, a returnable asset is used to ship glasses from a *glass producer* (owner of the asset) to a *distributor*. Instead of returning the returnable asset to the glass producers, the glass distributors might use them to further ship glasses to a *window manufacturer*. The window manufacturers might use the returnable asset to deliver windows to the final end customer, either directly or through a different *reseller*.

The table below summarizes the different stakeholders and roles considered in our use case:

Stakeholder	**Role**
Glass Producers	Deliver glasses either to distributors (directly) or to Window Manufacturers (through Logistic Provider). Own the returnable assets.
Window Manufacturers	Deliver windows to windows resellers either directly or through logistic providers. Might own their own returnable assets and use for the delivery. Or they might also re-use returnable assets received from glass producers.
Distributors	Receive glasses or windows delivered directly or through logistic providers together with returnable assets used for the delivery. Re-use the returnable assets to deliver glasses or windows further down to the distribution chain.
Windows Resellers	Receive windows from distributors and use received returnable assets to arrange their delivery to end-customers.
End-customers	Receive windows directly from windows resellers or through logistic providers which use glass producers and windows manufacturers returnable assets. Often dispose returnable assets or do not know to whom and how to return.
Logistic Providers	Move assets and returnable assets along the distribution chain.

The table shows how complex it is to keep track of all the relations involved into the handling of returnable assets.

The following graphic presents a simplified stakeholders map, highlighting returnable asset owners, the different chains of custody and how returnable assets move along the chain (dotted arrows) and those stakeholders (namely custodians, green circles) eventually responsible of returning the assets to their owners (purple circles).

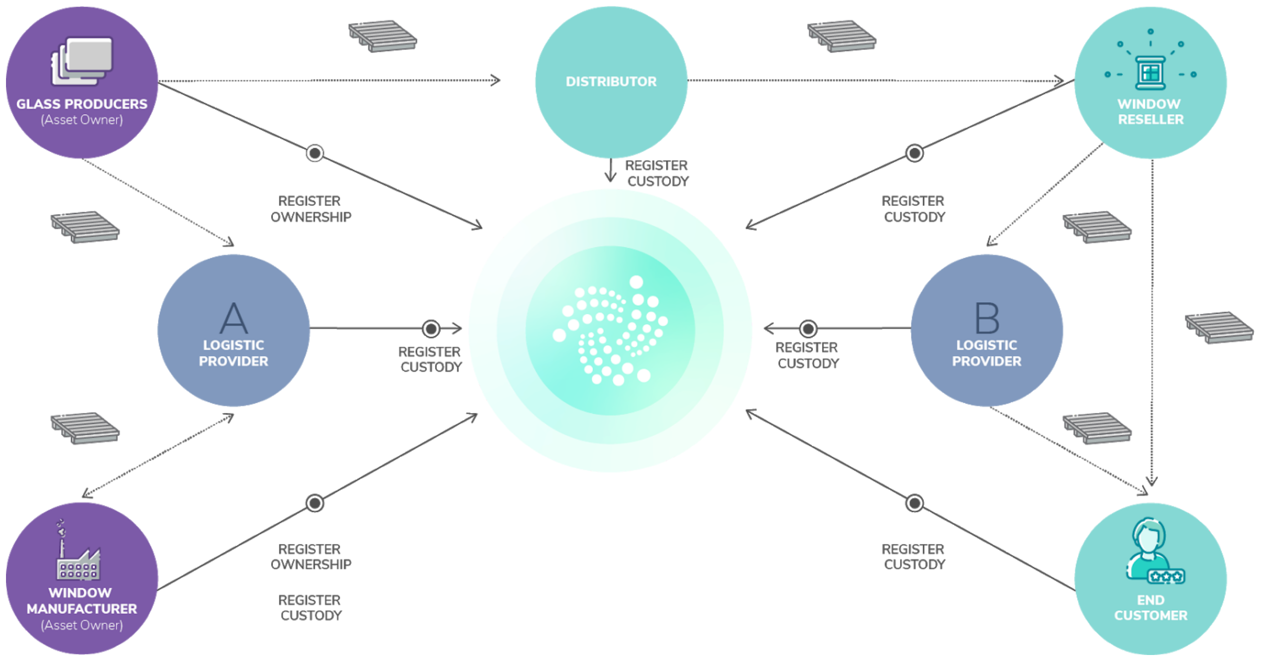

Fig 1: Returnable assets stakeholders map

The figure also shows the different actions they should perform when in contact with a given returnable asset.

Due to the lack of due diligence and complexity of this process, returnable assets are used within distribution networks and are seldom returned to their original owners.

Similar returnable assets and problems can be seen in other industries and value chains, such as the international flowers industry that also uses a variety of re-usable stands and pots. A solution to problem in the glass industry can be easily adapted to those markets and stakeholders too.

Therefore, there is the need for the returnable asset owners to track and trace their assets and to claim them back when these are considered missing. As various producers are circulating similar identical assets with their stakeholders, the problem is spread across the whole industry.

This requires a tracking system that allows for the collection of this information across a complex ecosystem of stakeholders. However, it is not possible to report custody-of-assets using a centralized database for the whole ecosystem as this will also reveal, to third parties, proprietary knowledge about different stakeholders, customers, and distribution chains. Moreover, it will be difficult to create a system able to centrally track all the possible interactions envisioned for a number of stakeholders not known apriori. The use of a distributed ledger provides a solution to these issues. In particular, using IOTA as a tracking layer ensures that only rightful owners will receive information of owned assets without any unauthorised party being able to gather or access any central recording of the full ecosystem.

Thanks to the permissionless nature of IOTA, *no trust is required for who runs the infrastructure* as this is spread across those independently running IOTA nodes. In addition to that, *no former knowledge of all parties willing to write information into the ledger is required*, thus simplifying the creation of *a track and trace system that can be used by all the different stakeholders and industry sectors as needed and with minimum integration and onboarding time*.

The solution we describe in this blueprint allows for the following:

- The owner of a returnable asset acquires the asset identity through a mobile app and an existing barcode or QR-code and creates a digital representation of her asset. The digital version of the asset (its *digital twin*) is recorded in an immutable way on the IOTA Tangle together with the identity of its owner;

- Once, as part of its use within the distribution channels, the given returnable asset is handed over to a different custodian, the new custodian uses the same or a similar app to retrieve from the IOTA Tangle the asset digital twin and to attach to it its identity as well as its location (if available). Information is again recorded onto the Tangle in order to further trace asset use;
- Change of custody is tracked as long as the asset and its custody is not returned to its owner or the asset is declared missing by its owner. In this last case, the asset owner can track the last asset custodian by retrieving it from the Tangle.

Business case

Misplace of returnable assets represents an economic loss for the asset owners. This also affects ability to fulfil producers next deliveries and is a waste of time and resources when owners try to recover their missing returnable assets.

Traditional track and trace of returnable assets has been so far unsuccessful for the following reasons:

- Economic value of returnable assets is not perceived by their custodians but only by their owners; instead returnable assets are more likely to be seen by their custodians as disposable assets;
- There is currently neither incentive for different custodians to help tracking returnable assets nor perceived responsibility in not doing that;
- Tracking returnable assets requires access to information stored in a number of proprietary systems belonging to the different custodians, for which the complexity to predict, map and integrate them exceed the perceived benefits.

The use of IOTA, a permissionless distributed ledger technology, provides a solution to seamlessly collect and share information about returnable assets, despite all of the involved custodians and without need to integrate any proprietary system. While doing that, IOTA can still guarantee access control of the collected information. The permissionless nature of IOTA Tangle, and the use of the 2nd layer MAM[1] protocol, fulfill these needs.

While this already solves the track and trace problem, in future scenarios the use of IOTA Token (and Qubic smart contracts[2]) could allow to create incentive to reward custodians participation to the track and trace activities, despite the country and the currency in which the assets are handled.

As a result, the following benefits are envisioned:

- Owners can track and request return of their returnable assets, thus saving money resulting from the need to buy new assets needed to fulfill their delivery needs. They can also save the time and the costs associated to searching for missing ones;
- Asset owners can better predict and plan shipments of their production by knowing exactly the number and location of returnable assets available to them;
- Custodians can easily track the returnable assets they handle, get rewarded and increase their reputation towards returnable assets owners, while saving costs associated to support owners' requests when assets are declared missing.

A scenario similar to the one described above is the one of the international container shipment, for which a solution similar to the one presented could be implemented and replicated.

[1] A MAM (Masked Authenticated Messaging) is a second layer communication protocol that allows to easily create and read data stream on the IOTA Tangle, guaranteeing integrity of the information and protecting its confidentiality through encryption. More info can be found here: https://blog.iota.org/introducing-masked-authenticated-messaging-e55c1822d50e

[2] Qubic: Quorum-based computations: https://qubic.iota.org/

Architecture

Running an open source project, like any human endeavor, involves uncertainty and trade-offs. We hope the architecture described below helps you to deploy similar systems, but it may include mistakes, and can't address every situation. If you have any questions about your project, we encourage you to do your own research, seek out experts, and discuss with IOTA community.

The presented infrastructure makes use of the IOTA Tangle and IOTA MAM[3].

The track and trace of a given returnable asset consists of a series of events. Because of this, registering the change of custody of a given asset, through the use of MAM channels, makes it easy to store the different custodian information onto the IOTA Tangle and associate it as an message in the same channel. Using MAM allows for encryption and protection of the shared information. Without using MAM, this could alternatively be done by issuing independent transactions to the IOTA Tangle, storing the required information for each change of custody related to a given asset. However, the architecture complexity of reconciling and linking all the information associated to a given asset would increase. Hence MAM was chosen as preferred solution design.

Figure below shows the main architecture components.

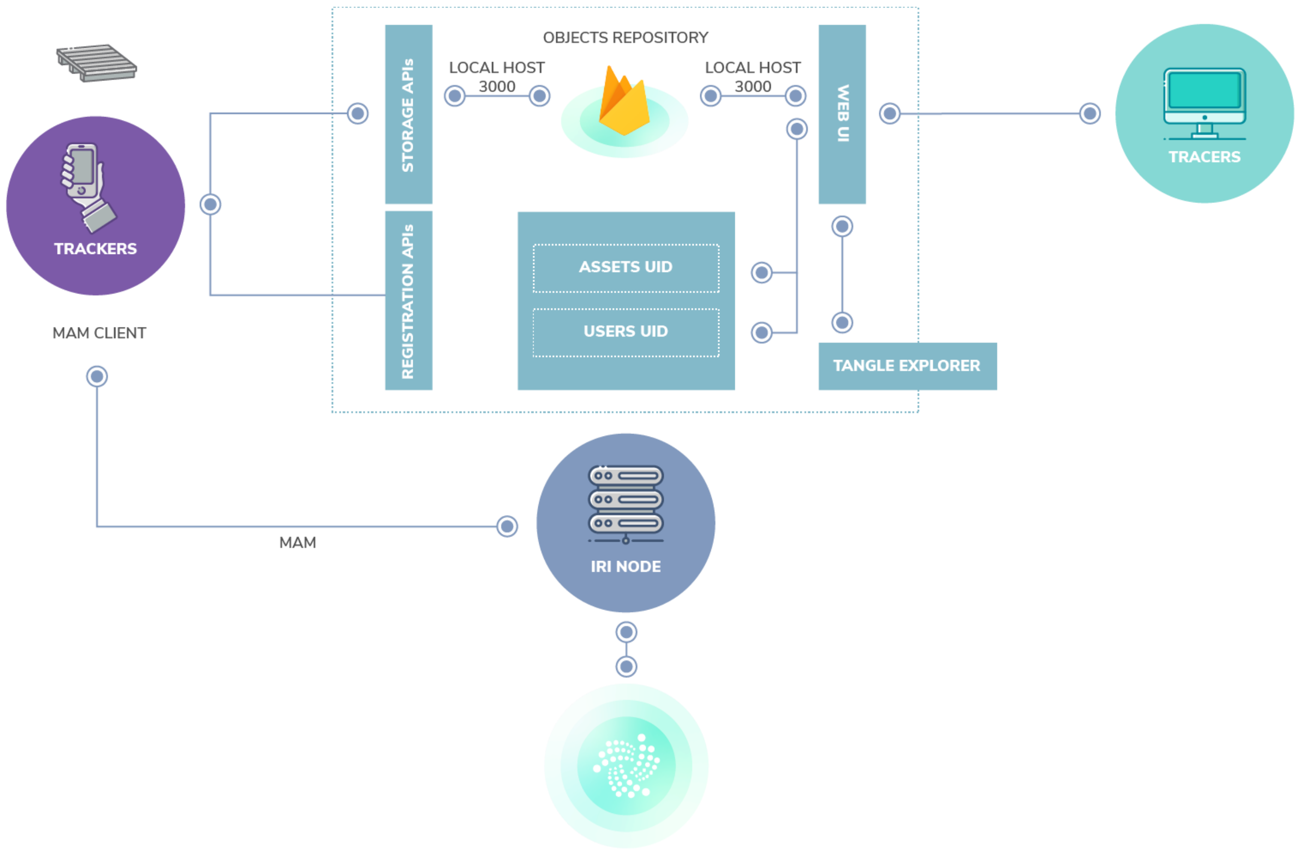

Fig2: PoC Architecture

A tracker interacts with the architecture to report asset ownership and change of custody. Authorized tracers

[3] MAM javascript libraries: https://github.com/iotaledger/mam.client.js

connects to the architecture to fetch information about a given asset, e.g. its last custodian.

A new IOTA MAM channel is created once a new returnable asset is first registered by its owner. A digital twin for the returnable asset is created with the following information: <assetUniqueID, assetOwnerID, assetCustodianID, location, time, status>. Required information is captured through a mobile app:

- assetUniqueID is captured through QR-code scanning and matched against an assetUniqueID server (e.g., GS1 SSCC), accessed through Registration APIs;
- information about the assetOwnerID is either inserted through the app or fetched from an external source (e.g., a registration server for the use of the app; company VAT registration number is used in this case);
- assetCustodianID initially match the assetOwnerID;
- location can be acquired by the mobile phone GPS (optional);
- time it is acquired by the device initiating the transaction (mobile phone or webapp);
- status can be either *in-use*, *returned* or *lost*. Initially it is set to *in-use*. However any different custom list of statuses can be defined.

The information is stored on the IOTA Tangle using the javascript MAM client library. This can either be embedded into the app or implemented through an external server (MAM Server), to which the app can exchange information using secure HTTPS REST APIs. For this blueprint MAM libraries are integrated directly into the developed app.

Tip1: An asset ownership is created and registered only once. While the asset custody changes over time, it is meaningful to use a MAM channel for aggregating the information related to the change of custody of a given asset. This allows to easily link all the different change of custody to the right asset by limiting complexity of the implementation. Moreover use of MAM also protects access to the relevant information related to each change of custody.

After creation of each MAM channel, a central back-end Object Repository is populated. The Object Repository is implemented as Firebase NoSQL database and deployed using port 3000. Storage REST APIs are provided to populate and update the Firebase DB with information related to the MAM channel associated to a given asset ID. Information stored in the Object Repository includes the root address of the channel, e.g., where this can be accessed on the IOTA Tangle and the cryptographic key needed for decrypting the information stored in the channel (named side keys), in case restricted MAM channels are created. The following tuple is created and stored in the Object Repository: <assetUniqueID, channelRoot, channelSideKey>.

The Object repository is either populated by the app or the MAM Server, according to the implemented model. Access to the Object Repository is managed by the given returnable assets owner, thus guaranteeing control on who can access and modify the information chain associated to a given returnable asset.

Tip2: Ideally every asset owner will extend the existing assets repository as part of their management system with capability to store the required information for access to the IOTA Tangle and MAM channel, instead of creating an external Object Repository. In the case of IBCS Group this was integrated into IBCS Tracker system (https://www.ibcstracker.com)

When the given returnable asset changes custodian, information about the new custodian is appended to the existing MAM channel. For that, a new MAM message is attached to the existing channel and the following information updated and stored on the Tangle: <assetCustodianID, location, time, status>.

In order to achieve this, the mobile app or the MAM Server of the new custodian needs to first retrieve the information related to the root of the MAM channel associated to the given asset from the Object Repository. This is done by using the assetUniqueID, as the primary key, which is obtained from the QR-code scanning or manually inserted. Information is then attached to the respective MAM channel and stored immutably onto the IOTA Tangle. For this, the two functions

```
createItem(eventBody, channel, secretKey, userId);
```

```
updateItem(eventBody, mam, newItemData, user);
```

have been implemented in order to respectively access and update existing MAM channel information (e.g. adding new messages to update the stored digital twin).

A Web UI (WUI) written in React implements APIs to access to the MAM explorer and to retrieve information, e.g. current custodian and location for a given returnable asset. Information on the Tangle are retrieved by accessing the required channel root address obtained from the Object Repository. A missing asset can be declared from the same GUI. *How to handle missing assets is out of the scope of this blueprint.*

The sequence diagram below recaps all the steps needed to track a given returnable asset.

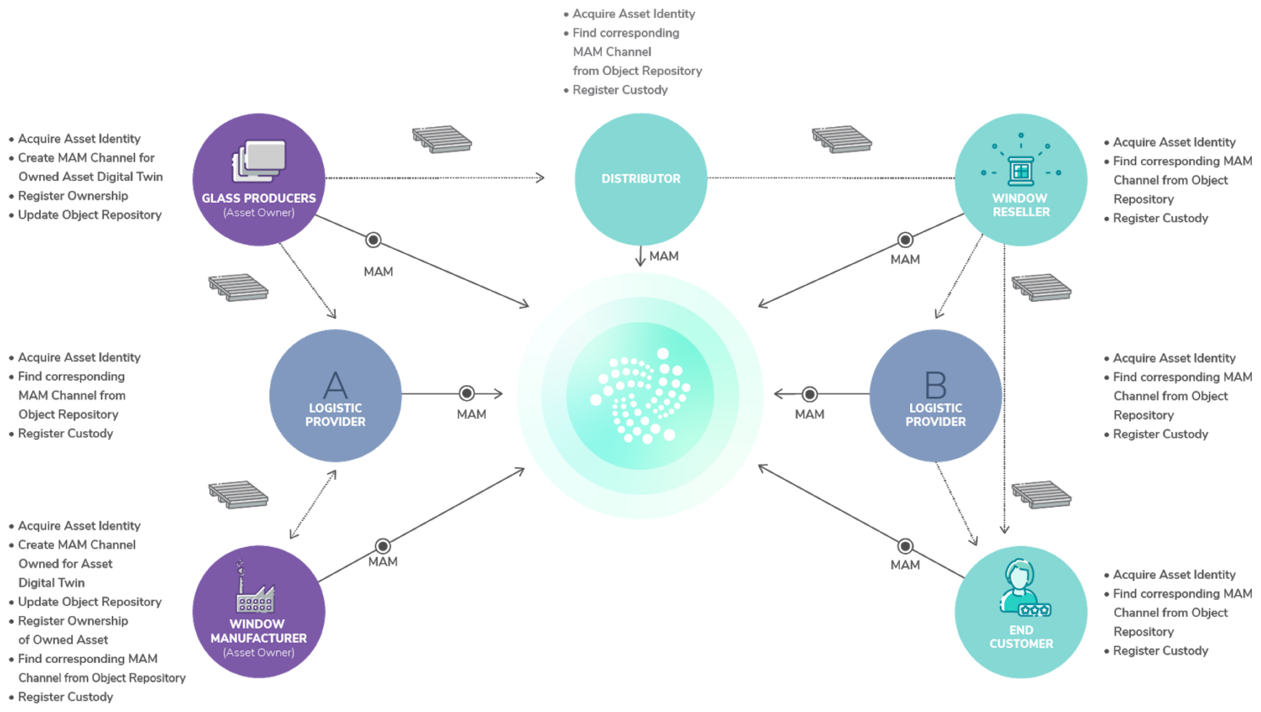
• Acquire Asset Identity
• Find corresponding MAM Channel from Object Repository
• Register Custody
• Acquire Asset Identity
• Create MAM Channel for Owned Asset Digital Twin
• Register Ownership
• Update Object Repository
GLASS PRODUCERS
(Asset Owner)
DISTRIBUTOR
WINDOW RESELLER
• Acquire Asset Identity
• Find corresponding MAM Channel from Object Repository
• Register Custody
MAM
MAM
MAM
• Acquire Asset Identity
• Find corresponding MAM Channel from Object Repository
• Register Custody
A
LOGISTIC PROVIDER
MAM
MAM
B
LOGISTIC PROVIDER
• Acquire Asset Identity
• Find corresponding MAM Channel from Object Repository
• Register Custody
• Acquire Asset Identity
• Create MAM Channel Owned for Asset Digital Twin
• Update Object Repository
• Register Ownership of Owned Asset
• Find corresponding MAM Channel from Object Repository
• Register Custody
MAM
WINDOW MANUFACTURER
(Asset Owner)
MAM
END CUSTOMER
• Acquire Asset Identity
• Find corresponding MAM Channel from Object Repository
• Register Custody

Fig3: PoC Sequence Diagram (actors and actions available)

The communication diagram below shows the different messages exchanged across the architecture components presented above.

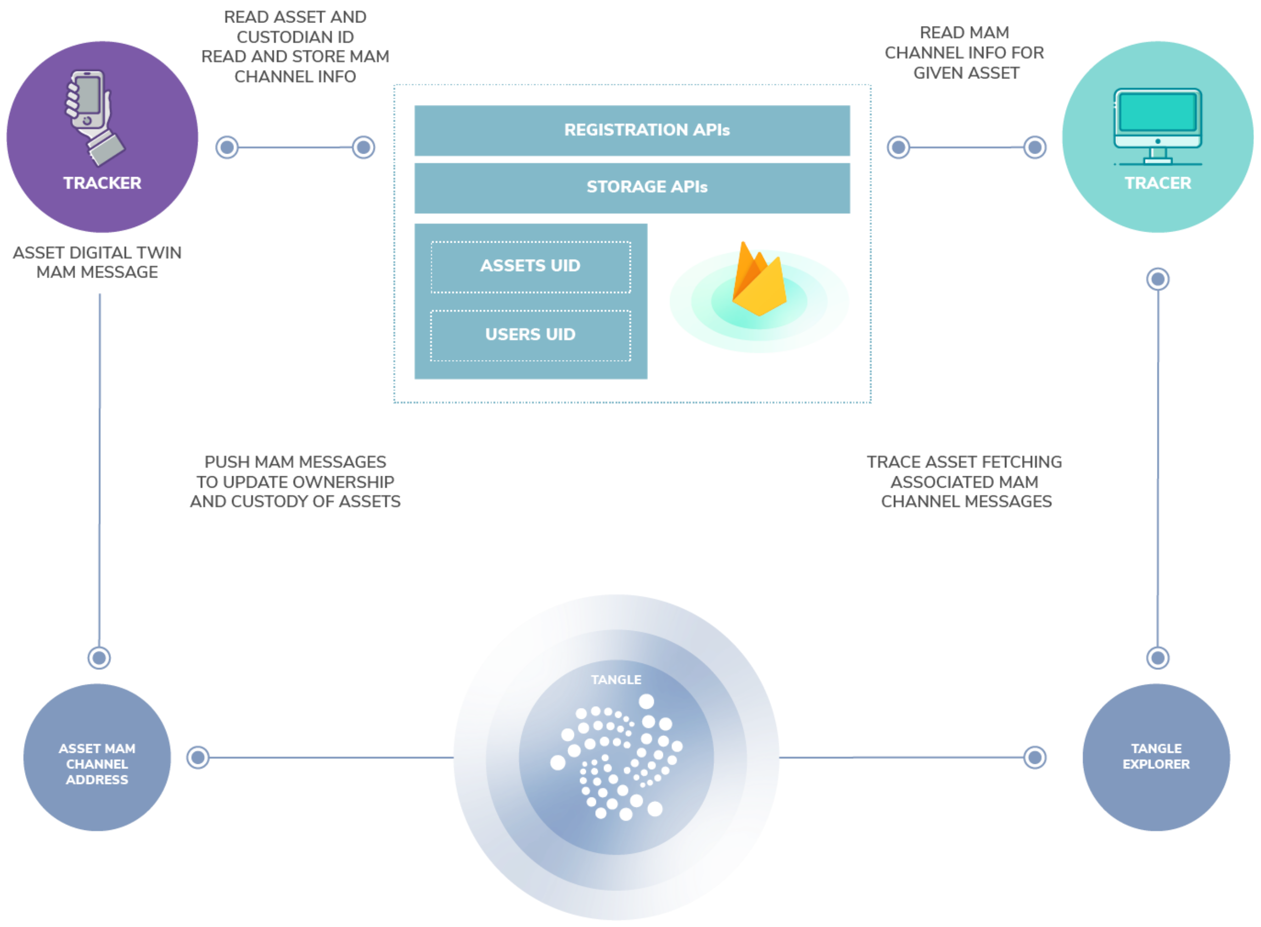
READ ASSET AND CUSTODIAN ID READ AND STORE MAM CHANNEL INFO
TRACKER
ASSET DIGITAL TWIN MAM MESSAGE
REGISTRATION APIs
STORAGE APIs
ASSETS UID
USERS UID
READ MAM CHANNEL INFO FOR GIVEN ASSET
TRACER
PUSH MAM MESSAGES TO UPDATE OWNERSHIP AND CUSTODY OF ASSETS
TRACE ASSET FETCHING ASSOCIATED MAM CHANNEL MESSAGES
ASSET MAM CHANNEL ADDRESS
TANGLE
TANGLE EXPLORER

Fig4: PoC Communication Diagram (messages exchanged inside the platform)

Details on the different components implementation is provided below, alongside with code snippets.

Data model/digital twin

A returnable asset digital twin contains the following:

```
{
        "data": [{
                "assetUniqueID": "string",
                "assetOwnerID": "string",
                "assetUserID": "string",
                "location": ["latitude", "longitude"],
                "time": "date",
                "status": "string"
        }]
}
```

IOTA building blocks

The tracker app will be responsible of creating and updating assets digital twins as MAM Channels and messages in order to allow tracking.

```
import Mam from 'mam.client.js';
import { isEmpty, uniqBy, pick, find, last } from 'lodash';
import { asciiToTrytes, trytesToAscii } from '@iota/converter'
import { createItem, updateItem } from './firebase';
import config from '../config.json';
```

First of all, we need to import the mam.client.js library.

Then before creating a MAM channels, we need to select the current IOTA network where transactions will be stored (provider). This could be the main IOTA Network or any dev network, such as: `https://nodes.devnet.thetangle.org:443`

```
// Initialise MAM State with IOTA provider
let mamState = Mam.init(config.provider);
```

For each new assets acquired, we need first to create the returnable asset digital twin (`createItemChannel`).

```
// create a new Item (Asset)
export const createItemChannel = (project, itemId, request, userId) => {
...

 const messageBody = {
        ...request,
        ...eventBody,
        time: null,
        location: null,
        assetUniqueID: null,
        assetOwnerID: null,
        assetUserID: null
      };
```

Before setting up the channel, it is recommended to set the channel mode to *'restricted'*. This allows to encrypt the payload (e.g., the information contained in the digital twin) of each MAM message associated to that channel and to guarantee access only to selected parties (`Mam.changeMode()`).

```
// create a new restricted channel
const createNewChannel = async (payload, secretKey) => {
  // Set channel mode for default state
  const defaultMamState = Mam.changeMode(mamState, 'restricted', secretKey);
  updateMamState(defaultMamState);
  const mamData = await publish(payload);
  return mamData;
};
```

We can then publish the information to the IOTA Tangle (`Mam.attach()`). Remember that IOTA uses Trytes so our MAM payload needs to be converted before sending it to the Tangle (`asciiToTrytes(JSON.stringify(data))`) and to create a MAM message (Mam.create()).

```
// store new messages for each new asset and for each change of custody
// Publish to tangle
const publish = async data => {
  try {
    // Create MAM Payload - STRING OF TRYTES
    const trytes = asciiToTrytes(JSON.stringify(data));
    const message = Mam.create(mamState, trytes);

    // Save new mamState
    updateMamState(message.state);

    // Attach the payload.
    await Mam.attach(message.payload, message.address);

    return { root: message.root, state: message.state };
  } catch (error) {
    console.log('MAM publish error', error);
    return null;
  }
};
```

Once a new MAM channel is created or an existing one is updated, we need to update the object repository. This can be done through the `createItem()` and `updateItem()` functions introduced above and described below.

```
export const createItem = (eventBody, channel, secretKey, userId) => {
  // Create item reference
  const itemsRef = getItemReference(eventBody.itemId);
  appendItemToNewUser(userId, eventBody.itemId);

  itemsRef.set({
    ...eventBody,
    mam: {
      root: channel.root,
      seed: channel.state.seed,
      next: channel.state.channel.next_root,
      start: channel.state.channel.start,
      secretKey,
    },
  });
};
```

```
export const updateItem = (eventBody, mam, newItemData, user) => {
  // Create reference
  const itemsRef = getItemReference(eventBody.itemId);

  itemsRef.update({
    ...eventBody,
    mam: {
      root: mam.root,
      secretKey: mam.secretKey,
      seed: newItemData.state.seed,
      next: newItemData.state.channel.next_root,
      start: newItemData.state.channel.start,
    },
  });
};
```

In the `updateItem()` function above, first the Firebase Object Repository is searched for an existing asset, through its itemId and subsequently information are updated with the new MAM channel or message details.

Deployment

The section below describes how to install and re-use the presented PoC.

Prerequisites

Programming: Javascript and React and basic knowledge of DBMS

For how-to deploy a Firebase server for the required PoC backend functionalities, please read here: https://firebase.google.com/

For how-to connect to an IOTA node, and sending transaction to the IOTA network using IRI, please read here: https://docs.iota.org/iri

For how-to to create MAM Channel and messages, using the IOTA MAM JS library, please read here: https://github.com/iotaledger/mam.client.js

How to use this blueprint

In order to reproduce this PoC there is no requirement to deploy dedicated HW.

More details and re-usable code can be found here: https://github.com/iotaledger/trade-poc

To deploy your track and trace web app, please follow instructions here:

https://github.com/iotaledger/trade-poc/blob/master/firebase_functions/README.md

To deploy your backend server (Object Repository) built in Firebase, please follow instructions here:

https://github.com/iotaledger/trade-poc/blob/master/README.md

Appendix

List of Abbreviations

IF	IOTA Foundation	
HW	Hardware	
SW	Software	
JS	Javascript	
DBMS	Database management systems	
IRI	IOTA Reference Implementation	the SW written in JAVA that allows users to become part of the [IOTA] network as both a transaction relay and network information provider through the easy-to-use [API].
MAM	Masked Authenticated Messaging	a second layer data communication protocol which adds functionality to publish and control access to an encrypted data stream, over the Tangle

Additional Resources

- IRI Repository - https://github.com/iotaledger/iri
- MAM eloquently explained - https://blog.iota.org/introducing-masked-authenticated-messaging-e55c1822d50e
- MAM Source code Repository - https://github.com/iotaledger/mam.client.js
- iota.js Repository - https://github.com/iotaledger/iota.js