2022

BloxMove technical review

FOR THE DUTCH MINISTRY OF INFRASTRUCUTRE AND WATER MANAGEMENT

ON BEHALF OF THE DUTCH BLOCKCHAIN COALITION,

AUTHOR:

NIELS KLOMP (SPHEREON)

nklomp@sphereon.com

https://dutchblockchaincoalition.org/en

version 0.2

Version	Date	Author	Remarks
0.1	13-03-2022	Niels Klomp	First version sent out for review
0.2	26-03-2022	Niels Klomp	Incorporating feedback and changes based upon additionally provided privately held documentation

Table of Contents

1 Executive summary

Most chapters have a short introduction and a conclusion. I advise any reader short on time to at least read chapters 1 and 2, then to read the introductions and conclusions of chapters 3 to 5. The other chapters are much more detailed and at a lower level for the interested reader.

This document is the published version of the review. It leaves out certain low level details at the code level as discussed with the bloxMove team, which are not necessarily meant for publication.

1.1 Executive summary

In 2021 the Dutch Ministry of Infrastructure and Water Management and the Dutch Blockchain Coalition did a Proof of Concept (PoC) with the bloXmove Mobility Blockchain Platform. The PoC focused on the use cases of re-usable customer onboarding and service roaming within the Mobility as a Service (MaaS) ecosystem. The PoC demonstration gained a lot of interest within the ministry and other stakeholders in the Dutch mobility sector.

As a next step bloXmove is planning the realize an initial production environment together with a limited number of shared mobility providers in the Netherlands. As a basis for a new open ecosystem, using the bloXmove blockchain platform. bloXmove and the involved mobility providers will finance this initiative themselves. However, they would like to involve the ministry for feedback and endorsement of their activities.

To decide upon potential next steps, the ministry has requested the Dutch Blockchain Coalition to perform a 4 day independent technical review to get a better insight in the maturity level of the bloXmove platform and verify if the upfront assumed design principles are implemented. The outcomes can be summarized as follows:

The current platform should not yet be considered production ready. Further development is required for a first production implementation with real customers and production data. bloXmove is aware of this but stakeholders considering participating should also be (made) aware of this.

The documentation of the current platform is available, but not open to anyone at present. The source code from large parts of the platform weren't available during the review. According to bloXmove the governance and cost model is still open for discussion. This results in a "black box" type of experience. bloXmove announced that it will open-source it's complete platform, with the exception of the Settlement Service by Q2 2022, using a liberal MIT license.

The ministry and the DBC considered a transparent, open-source solution crucial to facilitate a diverse public-private ecosystem with competitive mobility players. The use of closed source software components and the lack of open documentation prevent others in the ecosystem to efficiently evaluate the platform and collaborate with bloXmove. In a way it turns bloXmove in a centralized player and single point of failure. This is not the aim of bloXmove and therefore we advise the bloXmove team to further Open-Source their platform as soon as possible and prioritize their platform documentation. It seems the bloxMove team is aware and they stressed on multiple occasions that this is also their direction. An Open-Source solution makes it more attractive for others to collaborate and for the government to potentially endorse and support initiatives.

We appreciate the initiative and leadership that bloXmove is showing in this innovative space. We also very much appreciate the openness and willingness to participate with this review. The outcomes of this technical review should be considered as input for next steps in their roadmap.

Review outcome summary, based upon standards and potential

Overall level of maturity	Proof of Concept	
Type of blockchain	Public, or Private Permissioned	Depends on governance
Testnet available	Not yet	
Mainnet available	Not yet	
# of nodes main net	No mainnet yet	Indication of how decentralised the system is
License agreements	Large parts not yet open-source	Should become open-source in Q2 2022
Documentation		
Overall architecture	Not publicly available	Some sequence diagrams and swimming lanes are available. Private documentation is sufficient
Identities	Left out of scope on request	State is undetermined
Service Catalog Aggregator	Not publicly available	
Doorman component	Left out of scope on request	State is undetermined
Settlements & payments	Not publicly available	
Vehicle/T-box/Wallet integration	Not publicly available	
BLXM token	Depends on chosen governance	Depends on choices made
Cost model	Not available	Depends on governance
Governance	Initial presentation/high level available	Left open to decide
TOMP Gateway	Some documentation available	Some sequence flows provided
Fleetnode	Some documentation available	
Interop & technical standards		
DID methods	DID:ETH, DID:Web, DID:Ont	
Verifiable Credentials / JWT / JSON-LD	Yes /yes / no	
Signature suites supported	Unknown, no BBS+	
Interop eIDAS 1.0/2.0	yes	VCs are signed without eIDAS compatible signatures
CHAPI	No	
VC API	No	
OpenID Connect 4 Verifiable Presentations / OpenID Connect Verifiable Credenital Issuance / Self Issued OpenID Connect	No / no / no	
DIDComm (v1/v2)	no	

Presentation Exchange / Credential Manifests	No / no	
Risk on vendor lock in	Due to current unclarities there is a risk of vendor lock in	

2 Scope and depth

Performing an architecture and technical code review can be a pretty extensive process highly correlated with the size, complexity and quality of the underlying product. Given the scope is mainly about the performed PoC and getting a feeling about whether the platform provides a solid basis to continue, 4 days have been allocated to performing the review, which includes sessions with the bloxMove team and writing this document. The document should give enough insight into the state of the PoC and the overall bloxMove platform to the reader. If needed a more in depth review on certain aspects could be performed based upon conclusions. Given the amount of time available for this review, it could be that some conclusions need more nuance, that is also why the bloxMove team was involved in reading this document first. It doesn't necessarily changes the conclusions or outcome, but does serve to ensure that nothing was overlooked. If a longer review was executed more details in the code level would have been looked at, as well as whether the architectural components fit together properly and are secure. Having said that, 4 days gives enough details to make some assertions about the platform.

3 Source code

The source code of both the Fleetnode and the TOMP-GW is relatively easy to follow and properly structured into REST API Controllers, services containing the bulk of the logic and lastly common classes and interfaces.

3.1 Use of annotations

Swagger API annotations are being used to generate OpenAPI specifications from the classes and controllers. Some of the swagger data is not entirely in line with the actual implementation or seems to be copy/pasted from another endpoint, without updating for instance the description. This is however sporadic and something you encounter in a lot of software products at this development stage.

Validation annotations are also being used in places, to ensure data restrictions are enforced, like for example a property which is not allowed to be empty/undefined.

3.2 Separation of concerns (SOLID)

Some of the Service classes do seem to have more responsibilities than one would expect. Several examples of these are being highlighted in the Fleetnode chapter. Some refactor work probably is needed to increase the cohesion and reduce coupling. Noticeable examples are Authentication/Access token handling as well as VC/VP related code. Also some of the methods are relatively long, also indicating more separation/refactoring could help.

3.3 String literals

Throughout the code a lot of strings are being used in different places. Examples are URLs/paths, property keys and configuration keys. By having the same strings scattered across the code chances of bugs/mistakes increase, especially when strings are not consistently being updated. Therefore it typically is good to create constants out of the string literals so that an update only has to happen in a single location. The code could use some more constants, especially since the testing coverage is not optimal it seems.

3.4 Code duplication

There are some examples of code that has a high similarity and repetition, which probably should be refactored to prevent future bugs, but the amount of these instances is low and thus not a big issue.

Some of the best examples are the checkConsumerVerifiableCredentials in the rental helper service and also the vehicle registration item mentioned elsewhere. In the Fleetnode chapter an example is given that shows different behaviour as a result of duplication.

3.5 Overall code quality and other metrics

The cyclomatic complexity of most code is low, which is good. Some methods seem to be too long and would need to be split up. Some classes are also on the longer side.

Automatic linting of the code to check against common errors is setup. Continuous Integration is setup testing the software on source code changes.

3.6 Source code conclusion

Every software product can be improved from a source code perspective. There almost always is trade-off between time and money and development budgets, and other stakeholder requirements like time to market. The source code itself can be improved as mentioned, but it is in a state with a relative low amount of technical debt and written in a way that is easy to follow. As such the overall code itself seems to be of a relative good quality.

4 Testing

Testing is the process of automatically ensuring the software product operates and behaves as designed and thus as expected. It is an important aspect of providing, ensuring and maintaining software quality. There are multiple ways to ensure testing is in order. In this chapter the most important ways for a software product composed mainly of REST APIs are being discussed in the context of the review.

4.1 Unit tests

Unit tests are some of the lowest level tests available. They test an individual small piece of the total software product. The so called unit sometimes also referred to as module. This can be a single function or a set of functions, with one thing in common, they do not cross the boundary into other modules. Typically in software one module needs methods/services from another module. In that case the other module should have a fake implementation, which is called mocking. The reason this works is because the other module also should have its own unit tests as well as so called integration tests (see next sub-chapter).

By the looks of individual tests of the services, the available tests itself seem to have a proper depth and proper use of so called mocking of other modules. This basically means faking the other component, as the purpose of a unit test is to test your local component's code only.

Since I could not build the software myself because of the missing *`did-asset-library-core/nestjs`* library, I was unable to generate coverage data on the unit tests in place. Coverage data looks at the total amount of source code being tested, both in terms of files, lines and execution paths. By some manual inspections it seems that most services have some tests in place, but not every service method is being tested and some services are not really tested except for them being available. This is easily obtained from looking at which service methods are being called from REST controllers, with no test method in place in the associated test file for the service. The REST controllers are the contract to the outside world and thus the starting point for code execution. Overall it seems unit testing is focused on the more "important" functionality of a specific services.

Also service methods that do accept parameters do not seem to be tested against different values of these parameters. Different parameters typically lead to different execution paths within the code, and thus should be tested. The result is that the line coverage would increase if alternative inputs would be tested.

4.2 Integration and end-to-end tests

Integration tests are typically used to test across multiple modules in the same software product, ensuring they are working properly together

End-to-end (e2e) tests normally allow to test flows of the solution from beginning to end ensuring it operates as expected. Typically at least the high value flows, providing the bulk of the important functionality are involved in these types of tests.

There are some so called end-to-end (e2e) testing files in place. The e2e tests available however are only testing that the app itself is running and does not test separate flows unfortunately.

E2E testing is apparently being done by the bloxMove QA team, mostly in a more interactive way. The QA team is also responsible for the Postman collection used as part of the e2e tests. The postman collection is non-public at present and covers a quite extensive set of endpoints of the solution. The reviewer suggests to make these resources available to the public, accompanied by some documentation, so others can also properly test using data that is more in line with how production would look like.

4.3 Testing conclusion

The unit tests need to be extended to cover more service methods. Care needs to be given to have tests that handle different input parameters to test different execution paths in the code. Integration and end-to-end testing needs to be made publicly available with some documentation.

5 Documentation

Documentation is an absolute requirement for any software product. Documentation typically has different audiences and ideally documentation makes clear distinctions between these audiences. Both in terms of presentation and depth. Example audiences are system/technical administrators responsible for running the application, functional administrators, responsible for day to day operations from an end-users perspective. The end-users themselves and lastly developers, which can be external parties integrating software components and REST APIs, or developers of the software product itself. These are distinct types of developers that should be treated differently. The state of documentation is typically a good indicator of the maturity of a product, as documentation typically is lagging behind the actual software product features implemented and becomes more important at the point a product is being used and sold in production settings.

5.1 General documentation

The documentation provided publicly about the Fleet Node and the TOMP-Gateway is very light on details. It seems to be geared at software developers mainly and a small bit geared towards technical administrators. The TOMP-Gateway does provide some sequence diagrams that show how a mobility app, the bloxMove Fleet Node, the TOMP-Gateway and Transport Operator would interact. These diagrams where the most helpful available documentation during my initial review. The rest of the documentation is about the basic steps of running and testing the code, which is not extensive enough.

Having no access to the *`did-asset-library`* library meant it was impossible to build the software and test the setup instructions. Looking at the Continuous Integration (CI) file it seems that CI is done using Kubernetes and Helm charts. By the looks of these files, following the steps in the documentation would not have resulted in a running examples. Things like missing information on hostnames, changed DID methods, but also prerequisites like the IPFS setup and a local Ethereum ‘blxm’ network are missing from the documentation altogether.

The documentation in the non-publicly available Confluence is more in depth about the architecture, APIs and components. This documentation provides a better overview and most of that documentation ideally should be made available to the general public.

5.2 Source code documentation

The source code is not really documented. Source code documentation is targeting product developers and can be a double edged sword. It needs to be very accurate and in line with the code, otherwise it is better to have almost none at all and rely on accompanying documentation at higher levels. A lot of projects are somewhere in the middle, which means source code is not in line with the inline documentation (the Smart Vehicle ID Number (VIN) Service has some examples of these), which can become very confusing. As long as code is neatly written and has a clear separation of concerns, I am personally okay with not having too much source code documentation. In all cases you do expect to see some documentation/remarks in the code where for instance something unexpected is happening, or where there is a complex algorithm for instance. I did notice these types or comments in the code.

Although some engineers might not agree, the reviewer is okay with having this type of approach to source code documentation at this stage of the product.

The bloxMove team was very supportive in providing insights and explanations when things were not immediately clear from the documentation or code given. No “questionable” responses were given or answers that were not in line with the actual code. One thing to note though is the black box feeling as no insight/documentation was given about all the components involved in the whole platform.

5.3 SSI SDK

Some Documentation for the Apple/iOS SSI SDK was handed over. This SDK can be used in apps to create what bloxMove refers to as a native scenario. The SDK allows to manage keys and DIDs, as well as sign input data using these keys. The documentation provide pointers for the onboarding flow, as well as the rental process, where the SSI-SDK would be integrated into the iOS app of the provider. It has a mapping of methods to use cases and provides some pseudo code. The documentation of the SDK seems more detailed/complete than other documentation I have witnessed, but without giving actual source code examples probably it will not be enough for developers to successfully complete the integration without help. There also seems to be some slight mismatch between the SDK documentation and the actual implementation on the Fleetnode side. The confirm rental in the SDK expects a signature with the contract DID as input, whilst the Fleetnode expects a Verifiable Credential for instance. The getAccessToken endpoint in the Fleetnode however does have a header for this type of signature, but is not implemented. The SSI SDK needs to be refreshed a bit it seems.

5.4 Documentation conclusion

Conclusion is that the documentation currently would allow an experienced person, like an architect or senior developer to get their bearings in the bloxMove platform, but the publicly available documentation is too light to get any interested party up to speed easily. Especially documentation on the different architectural components involved like the Doorman components, Smart Contracts, IPFS/Storage, Service Catalog, Verifiable Credential Types, DID, Authentication/Authorization and their relationships are only present in the non-public Confluence environment. Some documentation for the SSI-SDK for IoS is available, which is a bit more extensive, but not enough to have an interested party integrate everything without help.

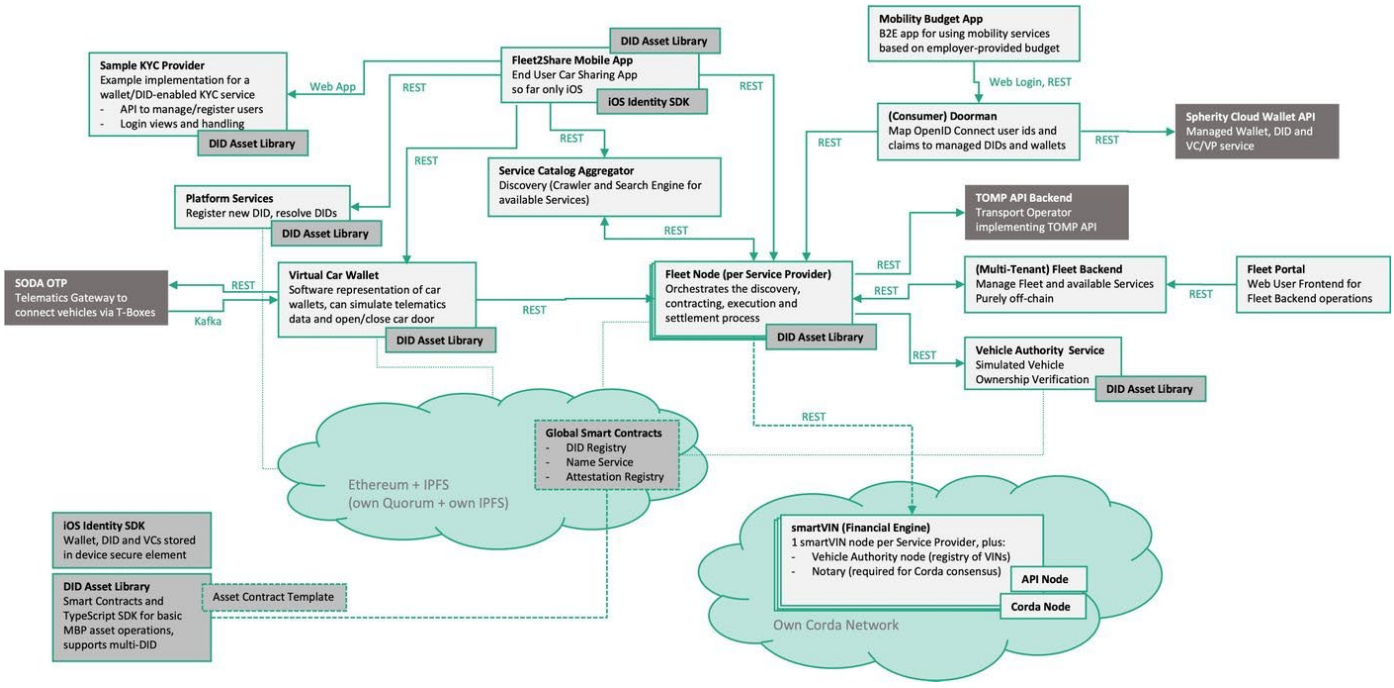
Given the answers provided by the bloxMove team, there is a bit more work to be done in this department, but the team should be able to produce comprehensive documentation given their knowledge of the platform and the already available internal documentation.

6 Architecture/Components involved

This review is mainly focused on the bloxMove TOMP-Gateway PoC and the so called Fleetnode. The Fleetnode is responsible for managing vehicles as well as renting vehicles either directly or B2B, which is referred to as backfill. Both software products are written in the Typescript programming language, which is a superset of Javascript. The applications are run using Node.js which is an Open-Source cross platform Javascript runtime environment.

The bloxMove TOMP-Gateway allows Transport Operators (TO) to integrate with the Fleetnode using TOMP APIs. This means the exposure of bloxMove concepts is minimized as the TO simply implements the TOMP interfaces. Only a subset of these interfaces is currently implemented though (service-catalog, booking creation/confirmation/end, and door-status).

The below overview picture is provided by the bloxMove team, which helps in putting some of the major components into perspective



6.1 Exclusion of the '*did-asset-library*' and `Doorman` component

Certain components have been excluded from review on as bloxMove deemed that potentially too much overlap might exists with work the reviewer's Company (Sphereon BV) is involved with.

By excluding the `*did-asset-library*` and the Doorman component, which form the SSI basis of the solution, obviously it becomes very hard to determine whether these libraries have been properly implemented. Especially since one of the objectives of the review is to ascertain whether proper SSI principles are being used. The Fleetnode code however uses the aforementioned library and its services and methods. This means that the Fleetnode reveals enough information about the so called interfaces of the methods it contains. Both input parameters and result values can be deduced

from the code. Most methods involved seem to be straightforward SSI methods for DIDs and VCs/VPs management, so the reviewer fails to see why these would be excluded. Especially in the light of bloxMove reaffirming it wants to make all the code available as Open-Source software.

The Doorman component is more opaque to the reviewer. This component is responsible for binding Authentication data with DIDs as part of an Identity Provider (IDP). This authentication happens outside of the Fleetnode/TOMP-GW and potentially has overlap with software the reviewer's company (Sphereon) is working on. To exclude this component from the review is a bit more understandable, although the same reaffirmation about making this available Open-Source has been given.

6.2 Did-asset-library

The *did-asset-library-core/nestjs*

The '*did-asset-library'* contains an Asset Service which is responsible for issuing and verifying Verifiable Credentials and Verifiable Presentations as well as DID management and resolution. Next to these responsibilities it can create attestations and retrieve data properties from assets like the Vehicle Id Number (VIN) from the vehicle asset DID. Basically it contains services and methods responsible for the core of the SSI work in the bloxMove platform.

The library also contains a Config Service, which as the name implies allows to get configuration values like settings and Fleetnode identities from configuration.

6.3 Fleetnode

Review remarks and findings about the Fleetnode code can be found in a separate chapter. This chapter explains the main services involved.

6.3.1 Vehicle management and registration

Vehicles managed and booked by the Fleetnode are referred to as assets. The actual API for the registration is the so called Vehicle Authority API, which is not part of the Fleetnode codebase. From the Fleetnode code we can however deduct that registration of a Vehicle requires minimal data, like a unique asset DID and proof of ownership with a signature that gets created from a message using the DID of the Fleetnode. The Vehicle Authority API also allows to check whether a VIN number exists or not. The Fleetnode uses this to ascertain that a VIN number is not issued yet and thus can be claimed during the registration of a new vehicle asset. As part of this flow involved vehicle registration wallet DIDs are added as involved parties using the so called Asset Service. This allows the respective DIDs to set/update data on the vehicle asset as well as sign signatures. The Asset Service is then being used to sign the proof of ownership and register that information

6.3.2 Service catalog

In the Fleetnode backend there is a notion of B2B interactions of Transport Operators (TOs) to make their vehicles available to one another by querying the Service Catalog using a channel filter. The Service Catalog is not part of the Fleetnode codebase and the reviewer does not have access to the

codebase to review its implementation. The service seems to be an application exposing a REST API that aggregates all assets of all TOs associated with the Mobility Provider running the BloxMove platform. The Service Catalog returns all available vehicles and their respective Fleetnode URLs.

6.3.3 Vehicle rental

The Vehicle rental service allows to request and confirm vehicle rentals by consumers as well as requesting and confirming the end of the rental. It also contains a method to update the door status of the vehicle. Lastly it allows to retrieve an access token to lock/unlock a vehicle. The input is the contract DID together with either a signature from the consumer with as digest the contractDID, or a Verifiable Credential. The consumer signature however does not seem to be implemented, leaving the VC as the only option. The response of that method is an Access Token response with an array of Verifiable Credentials, of which one is the VehicleAccessCredential. This credential expires after a configured amount of time and the subject has the consumer as holder, and contains information about the vehicle DID and contract DID.

The Vehicle Rental Service also has the concept of a backfill service. Basically this is a B2B service, in which a Fleetnode operator can use vehicles of another Fleetnode operator for its consumers.

6.3.4 Settlement

The settlement service is responsible for management of agreements (start, end and settle) as well as any assets/vehicles involved. It has contract data like prices, vehicle DID, consumer DID, terms and conditions, but also required user and business claims. Ending the agreement has usage data as its input. This data is about dates/times, mileage, geolocation and final price.

There are 2 implementations of the Settlement Service. One is a dummy implementation only logging the requests, the other is the SmartVIN Service, which can be enabled/disabled from the Fleetnode configuration. The actual SmartVIN REST API itself is not part of the Fleetnode codebase and thus not part of this review.

The SmartVin Service is the bridge to the Corda DLT. The Corda Identity for the FleetNode owner, is stored as part of the DID Document using the ‘alsoKnownAs’ field if present, which does pose a potential security issue. See chapter 8.6 for more details.

6.4 TOMP-GW

The TOMP Gateway allows the Fleetnode to integrate with Fleetnode Operators that either already have integrated with the TOMP API, or that plan to standardize on the TOMP API. All of the SSI heavy lifting is not done in the TOMP-GW component. That is delegated to the Fleetnode and its DID-asset library. Mainly the booking creation/confirm/end and update doorstatus endpoints are implemented, although at the service level a bit more is available. The code is currently publicly available at Github. It seems to be licensed using the liberal MIT Open-Source license from the package.json file. Given there is no License.md file present in the code, nor an explicit license selected within Github, the NPM package license currently is all to go by.

6.5 Ethereum Solidity smart contracts

The contracts are GPLv3, which as a more restrictive Open-Source license in terms of use in proprietary software, basically requiring that the software using a GPL library also becomes Open-Source according to GPL. What the exact impact is on software using the Application Binary Interface to interact with the respective contracts is left out of scope for the review. The team made clear that a more liberal MIT license will be chosen as the eventual Open-Source license.

6.5.1 DID registry

EIP-1056 compatible GPLv3 licensed smart contract that allows to check signatures, set or update the owner and/or delegate of the DID and set or revoke attributes of the DID. Seems to be based on the uPort's (Veramo/Consensys Mesh these days) ethr-did-registry project (https://github.com/uport-project/ethr-did-registry/blob/develop/contracts/EthereumDIDRegistry.sol) which oddly enough is more liberal in licensing (Apache2 & MIT), meaning the team made a choice to apply a more restrictive license. The team made clear that a more liberal MIT license will be chosen as the eventual Open-Source license though.

6.5.2 Asset

GPLv3 licensed smart contract for storing assets on chain. Allows to setup roles for assets (Admin, Controller, Signer, setData). Mainly delegates implementation to DID registry for attributes and owner. Allows to set specific key-values for the assets. The values can be either updatable or not. Also allows for delegation and delegation revocation.

6.5.3 AttestationRegistry

Stores VC hashes per topic per holder. The VC hashes act as attestations and are stored as raw bytes. The contract accepts byte arrays, meaning it is left up to the caller to ensure the actual hashes of VCs are being stored and not the full VCs themselves, as that could result in Personally Identifiable Information ending up on chain, which is certainly not desirable.

6.6 CORDA DLT

The Corda DLT is being used to settle agreements. Both the Fleetnode owner and the consumers/customers have identities on Corda, which can be stored in a field in their DID documents. No access/example of the Corda DLT integration is found. The SmartVin settlement service does provide some insight into the process though

6.7 IPFS

A local IPFS network is being used to store attestations (VCs/VPs) in a distributed fashion. No information or documentation was provided about it's usage.

6.8 Architecture conclusion

The architecture seems to be separated into multiple separate REST APIs in Micro Services style. This allows for independent development and eases in scaling a solution. As we will see in chapter 6, the publicly available documentation is lacking mostly, and access to all components has not been given to the reviewer, which makes the process of properly describing and ascertaining the architecture almost impossible. Luckily there is more internal documentation available which helped in forming an understanding of the platform as a whole. From the components that where available it seems that proper separation has been done, which should be easily scalable. There are some questions on how decentralized the architecture can operate, but that is mostly driven by the lack of insight currently.

7 Open-Source nature and Vendor Lock-in

The source code currently is not made available using an Open-Source license, with the exception of the Smart Contracts and the TOMP-Gateway. The latter has a Public Github repository, but is missing an explicit License file in the source code and has no advertised License in Github. The package file lists the source-code with a MIT license, which is a quite liberal Open-Source license, but it is unclear whether that was an explicit choice for the source code.

bloxMove has reaffirmed during talks that it wants to make the Fleetnode and other components available as Open-Source code at the latest by Q2 of 2022 using a liberal MIT license.

The Solidity smart contracts for Ethereum seem to be licensed under a GPLv3 Open-Source license, which is a less liberal license. Basically requiring an code incorporating GPLv3 code to also publish code as GPLv3. The choice for GPLv3 seems more explicit as the choice for MIT in the package file of the TOMP-GW component. Especially since one smart contract basically is a copy of an existing contract created by the uPort project which has a more liberal Open-Source license than GPLv3. This leaves the eventual licensing terms for all the component open. The team made clear that a more liberal MIT license will be chosen as the eventual Open-Source license.

Some components like the Service Catalog Aggregator, Settlement Service, Vehicle Authority Service, Corda integration were not available in source-code form to the reviewer. The same is true for the *`did-asset-library `* and *`doorman`* component because of overlap concerns with work of the reviewer's company. This means it is for the most part impossible to ascertain the status and License of these components, so all one can go by is the reaffirmation by bloxMove about the future license.

That does bring questions about what explicit license will be used for what components specifically, which directly impacts potential vendor lock-in. The reviewer suggests to seek more clarity with bloxMove around this subject, as open-source nature and the exact license(s) of the different software components heavily influence how open the code actually will be. The team made clear that a more liberal MIT license will be chosen as the eventual Open-Source license. That is nice of course, but at present it does mean that the components which have not been released yet, are to be treated as closed-source components until they are actually released.

8 Use of SSI standards and privacy

The Fleetnode software relies on the did-asset-library for doing the heavy lifting of the SSI work. As mentioned, the reviewer does not have access to this library. By the looks of the method calls into that library from the Fleetnode

8.1 DID methods support

Currently the platform supports the following DID methods:

- Web
- Ethr (Ethereum)
- Ont (Ontology)

The DID-ethr method is its main supported DID method. This DID method can store Decentralized Identifiers on both public and private/consortium run Ethereum networks. DID-Web allows management of DIDs by hosting the associated DID document using a webserver in a well-known location. That is especially useful for organizations that do not wish to integrate DLTs typically.

Given the use of Open-Source DID resolution packages, and the way the interfaces of DID management seem to be setup it is highly likely that other DID methods could be easily supported if desired.

8.2 DID subjects

DIDs seem to be used for different actors/entities within the platform. First of all there is the Fleetnode operator, which has its own (account)DID. Assets like Vehicles have their own DID, consumers (customers) have a DID and every contract gets its own DID.

DIDs for the Fleetnode and contract will not be a problem. Parties should be able to identify the same Fleetnode operator over multiple interactions over time. The fact that ever contract gets uniquely created receiving its own DID is also okay. That means that every rental has a new DID associated.

Vehicle DIDs being static from a Transport Operators perspective are okay for a PoC. Although this type of information can leak to other Transport Operators through for instance the backfill support, it does mean that Transport operators might be able to correlate more info than one ideally would hope for. Still, this isn't too big of a problem in a PoC as the car industry heavily relies on VIN numbers which are also globally unique. For production you really want to look at disposable IDs though.

A potential problem that needs careful consideration towards the future is the usage of DIDs for the consumers. If the consumer is using its DID across different Transport Operators, it means that all these Transport Operators share the same identifying key about a person across their systems. This is different from traditional systems where private keys are local to the system of the respective Transport Operator. In theory, Transport Operators could collude and share their data with each

other. Since in essence, their keys (DID) about the consumer are the same, they are generating data about the same subject, which now can easily be correlated and combined. For production use cases it probably makes sense to investigate whether Zero Knowledge Proofs and for instance techniques like Link proofs would make sense to prevent correlation by collusion as much as possible.

8.3 VC and VP support

The code makes heavy use of Verifiable Credentials and to some extend Verifiable Presentations. It seems that a deliberate choice is made to have Single-Use Credentials. These are Credentials that expose a single property about its subject. From a privacy perspective this means that the holder of the VCs typically can make choices about which VCs it will share as part of a VP, meaning it does not have to disclose too much information to a Verifier. Another approach is to use selective disclosure with so called Zero Knowledge Proofs. This brings additional privacy options, like not even having to disclose the holders ids, but at the cost of the technology being less mature. The bloxMove platform currently does not support selective disclosure using for instance Zero Knowledge techniques.

8.4 User/Business Requirements

Certain requirements for use cases are implemented by the consumer or other party needing to possess certain types of credentials issued to them. The best example is the ‘Over 18’ requirement to drive a car for instance. Basically It means the VC is issued to a holder after a KyC check has been performed by the issuer on the subject. From that point on the subject sends in that respective ‘Over 18’ VC whenever needed and only as part of a VP, to prove the subject binding. The code seeks whether these VCs are present in certain use cases, like renting a car.

A solution like that works and is not reliant on for instance a Zero Knowledge solution which would be able to derive these types of requirements from for instance the data of birth of the subject, without disclosing that actual date. The reviewer does suggest to the bloxMove team to start using the DIF Presentation Exchange specification whenever a Verifier would like to pose certain restrictions on the VCs it would like to see.

8.5 VCs are sometimes used differently than one would expect

The structure and usage of VCs and VPs around the rental process sometimes seems different from what you would expect initially.

Starting a rental request as a consumer uses VPs as one would expect. To bind the VCs that have been issued by a KyC provider, containing claims like for instance ‘over 18’ and ‘has a driver license’. There is a proper usage of the subject being the consumer and the issuer being the KyC provider. Then the consumer signs the VPs when starting the rental request. The request contains the car (DID) and a packageId and returns a uniquely created rental contract DID to the consumer. Instead of only returning the rental contract DID, one could also have expected that the Fleetnode would issue a Contract VC at this point, as the contract was created by the Fleetnode.

What happens next is that the consumer issues a VC containing the contract DID as subject and the consumer itself being the issuer. This means that the proof contains a signature of the issuer/consumer. The Fleetnode might be able to correlate the initial subject id of the start rental request with the issuer of this VC, but if a VC would have been returned in the previous step this process would be more explicit, as the contract would have been bound to the consumer already. All that is needed is sending in a VP instead of a VC as the payload to confirm the rental. The Fleetnode than has an easy job of verifying the subject being the holder and knows that the issuer was itself to begin with. Using a contract DID as subject doesn't seem to most logical solution. Unfortunately the reviewer does not have access to the actual smart contract code of the rental contract, but from the test code it seems that both the consumer DID as the vehicle DID are present in the contract. It could be there is some logic in the smart contract that warrants this approach however.

The confirm step doesn't return data. It relies on the returned HTTP status code to denote a successful confirmation. Issuing a successful booking as a VC could be added here.

After getting access tokens to unlock/lock the vehicle the rental end request will be initiated by the consumer. This is in the form of the consumer issuing a VC with again as subject the contract DID following the same pattern as the start. This also alternatively could have been a VP, creating a more direct binding. The benefit of this solution would be that the Fleetnode could actually revoke the "Rental VC" as part of the VP, which to the reviewer seems to be a more natural flow. The response contains usage data.

Lastly there is the end rental confirm step, where similar to the start confirm step the consumer sends in a VC with itself as the issuer and the contract DID as the subject.

By making the consumers issuers and also having them control the contract DIDs as subject it automatically means that additional security checking needs to happen on whether the rental contract really includes the consumer (DID). The Fleetnode is handling creating and returning the contracts and their associated DIDs. One would also expect the Fleetnode to issue VCs with that information contained as subject data, with the consumer as subject id. Then a simple verification of the VPs and a check for holder is subject is enough to ensure nothing bad can happen. The VC in the VP is issued by the Fleetnode after all and can be cryptographically proven.

Unfortunately I do not have access the *`did-asset-library`* to properly asses the security impact of this approach, but it is hard to imagine that this has no impact on the overall security of the solution as the self-asserted VCs basically are only proving possession of the VC after issuance. The security seems to come from the accountDID being the same linking pin between requests and offers. Obviously offers can only come from the fleetNode owner's accountId, so these are cryptographically protected and thus provide some level of protection.

A more common approach would have been to issue VCs from the fleetNode owner, optionally mixed in with self-asserted VCs, where VCs from the fleetNode owner either have:

- The consumer as the subject ID and then the contractDID as another field in the subject object
- Not so common: Multiple subjects, one id for the contract DID and one for the consumer/customer DID as a 2nd subject

Then the consumer would be handing over VPs with a proof (signature) including a challenge and domain.

From the Fleetnode's perspective this would mean it can always ensure to accept VCs from itself exclusively, or other trusted fleetnodes as it inherently doesn't trust consumers perse. Next to that it knows that the VP comes from the actual consumer it handed out the VC to, as there is a check in the VP against the subject of the VC(s), by means of the proof. Lastly you get automatic replay protection.

8.6 Potential security related issues

There are 3 potential security related issues that stood out, of which the first one seems to come from the setup of VCs. Please be aware that the first two issues might be properly handled in the code, but the reviewer believes the first issue by its design now crosses multiple boundaries and could have been contained to a simple cryptographic proof, making the chances of an exploit happening a lot lower.

1) There are constructs where a VC is being checked, like for instance on the end of a rental. It is unclear why these aren't Verifiable Presentations (VPs). You would expect a VP rental request from the consumer and then the issuance of a VC during start of a rental by the fleetnode owner, followed by a VP end rental request from the consumer during the end of a rental. This ensures that the VP really was signed by the same party as the subject DID of the rental request (VC).
   Right now the security comes from handing in a valid VC and the end rental request containing the same DID as the subject DID. It means that both the contract DID and the contract needs to be inspected for a proper match of the consumer. It also means the issuer signature of the end rental request needs to be checked together with the issuer being the consumer in the smart contract, otherwise an attacker only needs the issued start rental VC to end the rental. He has a hold of the VC, thus knows the contractId as it is the VC subject id. If the VC falls into the hands of someone else, this person would be able to end the rental, without having to prove control over the DID. Changing the setup as described earlier means simple cryptographic checks makes these types of potential pitfalls impossible.

2) The Corda Identity for the FleetNode owner, is stored as part of the DID Document using the 'alsoKnownAs' field if present, which does pose a potential security issue, especially from the DID controller's perspective. Normally a DID is in full control of the owner, called the controller. This is one of the key characteristics of Self Sovereign Technology. Thus relying on any field other than public keys is dangerous as it allows the controller to update their DID Document and change the respective value into a Corda Identity of another entity for instance. This means that security measures need to be in place elsewhere to prevent this from happening. After some discussions with the team, the reviewer is properly convinced

that this is very unlikely to happen, given the Corda Node is running next to the Fleetnode of a Transport Operator. In turn the Corda Node is protected cryptographically and not accessible by others. So even if one would use the Corda Id of another party, the cryptography and access would not be possible.

3) The cryptographic check on submitted VCs happens after already performing some other actions, like for instance retrieving the rental data using the rental contract DID. One would expect that check on VCs/VPs to have happened first to ensure that the user is actually in possession of a valid credential to begin with. It is good practice when authentication and/or authorization is involved, to do these checks as early as possible, to not potentially disclose information to an attacker, like for instance whether a contractId is valid or not.

9 Extensive answers to predefined questions

Is the overall architecture clear and documented. That includes both the TOMP API/GW and bloxMove platform itself . If not what is missing?

The documentation is available. Except it currently is mostly closed-off from the outside world, residing on their Wiki/Confluence. The depth of the documentation that is publicly available is mostly superficial. The not-publicly available documentation has more depth and also contains Architectural documentation, API design and usage/integration documentation.

The architecture is properly documented, but not publicly available. There are multiple components, REST APIs, ledgers and smart contracts involved. The documentation is geared towards building and running the Fleetnode. The architecture becomes clearer from the non-public documentation and by looking at the source code and the test available, since some systems and components are being mimicked (mocked) to properly test small units of the code.

The public Fleetnode documentation is not sufficient currently

How Open-Source is the solution, or parts of the solution and what is the intent and planning if parts are not Open-Source (scope of the Open-Source nature). What type of license is involved?

Most code currently is not available with an Open-Source license. The exceptions are the PoC and partial TOMP Gateway which is available with a liberal Apache 2 license, and the smart-contracts sent via e-mail by the bloxMove team, which are available with a restrictive GPLv3 license. The team made clear that most code should be released as Open-Source using a liberal MIT license in Q2 2022, with the exception of the Settlement Service.

The DID Asset library, which does most of the SSI heavy lifting and the Doorman component, which bridges authentication with VCs, was explicitly left out of scope on request of bloxMove, given the potential overlap with the work reviewer is doing. Then there are other components that the reviewer did not have access to, like for instance the Service Catalogue aggregator and the Vehicle Authority service.

The above basically means that a lot of code and components currently are not available as Open-Source software. The team did stress on several occasions that almost all code will go out as MIT. Some parts of the pure crypto solution, which includes the BLXM token might be licensed GPL, but that is not part of the envisioned Dutch ecosystem.

The above does mean that currently most code is not Open-Source and thus inherent risks are involved if the different components, the license type and the timeline would not be put in writing.

If there are Closed source components, are they well documented, and could they be replaced if there would be a need (Vendor lock-in)

By extension of the above, the answers is: yes there are currently closed-source components, including the Fleetnode itself as well as other external REST APIs, like the Service Catalog Aggregator, Vehicle Authority, Virtual Car Wallet and the DID asset library. They are documented well, but this documentation is not available to the public. The bloxMove teams will open-source all of their current offering with the exception of the Settlement Service. This service is described well enough to be replaced by another solution if parties would see a need.

The current reality is that most components are not made available yet with an Open-Source license. Meaning a lot of code is closed-source. See above.

By the looks of the testing code a lot of these components would be relatively easy to replace with Open-Source components if the respective components would not be made available Open-Source. The biggest exceptions it the Doorman components, as from the testcode is does not immediately become clear how this component exactly works.

Are open standards being used where applicable (Vendor lock-in)

For SSI, Decentralized Identifiers (DIDs) and the Verifiable Credential Data model are being used, which are both W3C specifications. Currently it seems like no SSI transports like DIDComm, CHAPI, OpenID, VC API are being used, as the VCs and VPs are being used as payloads in the requests and responses of the REST API (which is fine). The DID registry on Ethereum is based on the well-known uPort implementation which in turn implements an Ethereum Improvement Proposal (which is an Open Ethereum standard).

What is the overall Technology Readiness Level at this point? How mature is the solution?

The current code seems to be in a Proof of Concept state, close to a Minimal Viable Product (MVP) by the looks of the Fleetnode code. More publicly available testing and documentation are prerequisites though. Also some of the code is clearly written to take the easy route, meaning geared towards a PoC or MVP at best.

In order to determine the Technology Readiness Level a proper Technology Assessment would need to be conducted. We will follow EU readiness levels, based upon an assessment of currently available information. The levels are described for instance in https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf are as follows:

- TRL 1 – basic principles observed
- TRL 2 – technology concept formulated
- TRL 3 – experimental proof of concept
- TRL 4 – technology validated in lab
- TRL 5 – technology validated in relevant environment (industrially relevant environment in the case of key enabling technologies)
- TRL 6 – technology demonstrated in relevant environment (industrially relevant environment in the case of key enabling technologies)
- TRL 7 – system prototype demonstration in operational environment
- TRL 8 – system complete and qualified
- TRL 9 – actual system proven in operational environment (competitive manufacturing in the case of key enabling technologies; or in space)

The Fleetnode and TOMP GW are certainly at least at level TRL 3. The Fleetnode and other APIs, minus the TOMP GW, probably are at level TRL 4 to TRL 6. This is devised based upon the below findings.

The current code seems to be in an advanced Proof of Concept state, towards a MVP. Lack of public documentation, tests, licenses not being open-source yet and not having access to all the components are clear indicators of the platform not being production ready. This makes TRL level 7 or higher impossible at present.

Is privacy of travellers secured?

Yes and no:

Yes, in the sense that SSI is being used, meaning that attributes about the traveller's identity are being stored in the device of the traveller itself. This obviously has a benefit from a privacy perspective. The solution uses single-use VCs, meaning that a single identity attribute is represented by a single VC. This gives the traveller a lot of flexibility when attributes need to be disclosed as a Verifiable Presentation, at the expense of potentially being more problematic if these VCs would also be used in a more generic type of SSI wallet in the future. The team could look for so called Zero Knowledge types of proofs in the future, to allow for more flexibility and privacy, allowing for

selective disclosure. Especially with current Credential Manifest issuance specification and Presentation Exchange for VC definitions and validations in the hands of the traveller.

No, in the sense that the solution introduces a single DID for the traveller currently. A DID is a unique identifier. Normally a single system of a company uses a unique identifier (primary key) to track a single user or asset for instance. In this case the DID acts as that identifier. With the big difference, that this DID is used cross company and Transport Operator, meaning you have a unique value to track the traveller across all of them. In the future this needs to be addressed with for instance Zero Knowledge Proofs / link secrets or similar privacy preserving techniques.

Is anything needed for eIDAS interoperability?

DIDs are being used as pseudonyms for travelers, since they uniquely identify a single traveler, even across Transport Operators. This is something to be aware of. From the eIDAS perspective not too big of a problem, same for the right to erasure. It is not something that is really applicable from the bloxMove platform's perspective. As it is quite easy to prove that the information being gathered is necessary for the Transport Operator to operate. The problem might be more outside of the scope of the platform with TOs keeping track of the data for other purposes.

Like most projects non ETSI signatures are being used, meaning the signature scheme is not in line with eIDAS. This is however an SSI industry problem currently and has to do with the different approach SSI takes from more traditional Certificate Authority based certificates.

The attestations which also act as an audit trial are stored using the hashes of VCs/VPs and thus are unique across invocations. The actual VPs/VCs are stored on IPFS, which means that the data could be made inaccessible (right to erasure).

That does bring a final remark in terms of eIDAS. Data needs to be kept according to specified terms and timelines for defined purposes. This also means that data needs to be purged after it has exceeded these terms. Although it seems technically possible from the platforms perspective, the reviewer has no insight into whether it is actually implemented or not.

Is the solution scalable

By the looks of the design of the Fleetnode it should be. Currently it seems that the respective Corda/Ethereum ledgers and potentially IPFS could be the bottleneck in the architecture. Most of the code should be easily scalable horizontally and vertically by the looks of it.

What ledgers, DID methods and SSI technologies are being supported? What would be needed for interop with other ledgers/DID methods?'

Currently Ethereum is being used for the smart contracts around DIDs and asset registration. Other EVM compatible blockchains exists and the smart contracts are straightforward, so they should be easy to port. Next to that Corda is being used for settlement. Corda is a logical choice. If needed an alternative solution could be chosen, even without having seen the actual integration.

3 DID methods are mainly being used. DID web, which used no blockchain at all and allows for publishing of Public keys/Verification Methods using a webserver. DID onto, from the ontology project. It is not entirely clear why this method is supported and what benefits it brings next to interoperability with that particular blockchain. The main blockchain based DID method is Ethereum, with its DID ethr method.

It should be very straight forward to integrate other DID methods. Although the DID integration is part of the DID asset library to which the reviewer did not have access. The process of DID resolution is very easy. The process of DID registration is really dependent on the respective DID method, but typically also not very complex.

Are the TOMP API and DID/VC standards correctly implemented?

Only a subset of the TOMP API (service-catalog, booking creation/confirmation/end, and door-status) is implemented. Basically it maps everything to the Fleetnode which handles the SSI internally. The implementation seems straightforward and clearly is at a PoC level at this point (as one can expect, given it was executed as a PoC).

The DID/VC standards are probably implemented correctly. I do see use of single-purpose VCs to maximize privacy and the use of Verifiable Presentations to allow for holder binding of the VCs, proving that the holder owns the keys associated with the DID to which the VCs were issued. Given the DID Asset library was left out of scope, there is not much more to tell about it, with the final remark that some API requests/responses might seem a bit different than one would expect. This is mentioned in chapter 8.5.

Is there enough insight into the workings of the Smart contracts and settlement components?

There is enough insight into the working of the smart contracts, as the source code was shared. As to the settlement components, the answer is no, as the code is not shared. The SmartVIN service in the Fleetnode does give a small bit of insight, but not enough.

Is there enough insight into the "Doorman" component?

Simple no. No code was available to the reviewer, no architecture and it is not really deducible from testcode. The bloxMove team will provide a blackbox description of the component though.

Is the "discovery/service catalog aggregation" properly described and could it server as a central component in the sector?

No code was provided for the Service Catalogue Aggregator. It should act as a low level crawler component. From the Fleetnode code which has some integration with the Service Catalogue aggregator is becomes more clear what the aggregator is about, but with lack of insight at present all that can be said is that it currently is not a solution that can act as a central component for the sector.

What is the role of the BLXM token in the architecture and solution?

From the code perspective at present there is no role it seems.

Gathered from the team, they ultimately want to have the BLXM token on the public CELO mainnet, as the primary means to pay for services. That doesn't exclude parties paying in fiat/money, it simply means that for every fiat transaction a BLXM transaction will also be provided by a third party.

For a private/permissioned network the BLXM token is not necessary perse. It really depends on the governance chosen. The working assumption by bloxMove is that for the Dutch collaboration either a private blockchain or a hybrid public/private blockchain without BLXM token would be needed. It would then be governed by Dutch institution(s) as opposed to the governance of CELO. If there is interest in a token approach then bloxMove is open to discuss.